Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

О. Ю. Горобець, С. В. Горобець, К. Ю. Хахно

Мова Python
для інженерних та наукових задач
Підручник

Затверджено Вченою радою КПІ ім. Ігоря Сікорського
як підручник для здобувачів ступеня бакалавра
за спеціальністю 162 «Біотехнології та біоінженерія», магістра за спеціальністю за спеціальністю 162 «Біотехнології та біоінженерія», бакалавра за спеціальністю 104 «Фізика та астрономія» та магістра за спеціальністю 104 «Фізика та астрономія»

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2024

УДК 004.43 (004)
Г 19

Автори: *Горобець Світлана Василівна*, д. т. н., проф.
Горобець Оксана Юріївна, д. ф.-м. н., проф.
Хахно Кирил Юрійович

Рецензенти *Мамілов С.О.*, к. ф.-м. н. старший науковий співробітник, завідувач лабораторії «Лабораторія біосенсорів №12» Інституту магнетизму НАН та МОН України
Дереча Д.О., к. ф.-м. н. старший науковий співробітник, завідувач лабораторії «Лабораторія нанокристалічних структур» Інституту магнетизму НАН та МОН України

Відповідальний редактор *Голуб Наталія Борисівна,* д.т.н., професор, завідувач кафедри біоенергетики, біоінформатики та екобіотехнології КПІ ім. Ігоря Сікорського

Гриф надано Вченою радою КПІ ім. Ігоря Сікорського
(протокол № 4 від 01.04.2024 р.)

Горобець О. Ю.

Г 19 Мова python: для інженерних та наукових задач [Електронний ресурс] : підруч. для здобувачів ступеня бакалавра за спеціальністю 162 «Біотехнології та біоінженерія», магістра за спеціальністю за спеціальністю 162 «Біотехнології та біоінженерія», бакалавра за спеціальністю 104 «Фізика та астрономія» та магістра за спеціальністю 104 «Фізика та астрономія» / О. Ю. Горобець, С. В. Горобець, К. Ю. Хахно ; КПІ ім. Ігоря Сікорського. – 1-ше вид. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2024. – 277 с.

В підручнику викладено основи мови програмування python, а саме базові оператори, рядки, списки, кортежі, множини, словники, функції та класи. Також в підручнику викладено найбільш популярні методи пакетів matplotlib, numpy, opencv, scikit-learn, pandas, Biopython, які необхідні для розв'язання інженерних та наукових задач та графічної візуалізації результатів. Підручник містить приклади розв'язання задач аналізу зображень, регресійного аналізу, апроксимації та окремих задач біоінформатики. В тому числі наведено приклад розрахунку похибки результатів вимірювання з застосуванням методів пакету numpy. Окрему увагу приділено історії розвитку мов програмування та парадигмі об'єктно-орієнтованого програмування. Підручник призначений для здобувачів ступеня бакалавра та магістра за спеціальностями 162 «Біотехнології та біоінженерія» та 104 «Фізика та астрономія». Матеріал підручника буде також корисним для студентів спеціальностей природничих наук, слухачів курсів, інженерів, викладачів та всіх, хто вивчає мову програмування python.

УДК 004.43 (004)

Реєстр. № П 23/24-024. Обсяг 13,7 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

Зміст

Вступ

Сучасний світ не можливо уявити без інформаційних технологій, які увійшли у всі галузі. Професійний біотехнолог, фізик, інженер наразі має знати та вміти використовувати сучасні мови програмування. Лідером серед таких мов є мова python, чи не найбільшими перевагами якої є відкритість вихідного коду (безкоштовність та доступність для подальших покращень), об'єктна орієнтованість (заснована не на функціях, а на об'єктах з певними атрибутами й методами), високий рівень (зручність для програміста), загальне призначення (можливість використання для створення будь-яких програм). Для здобувачів вищої освіти в галузі природничих наук безкоштовне використання мови python є особливо важливим через те, що спрощує публікацію результатів наукових досліджень студента в закордонних фахових періодичних виданнях при роботі над бакалаврськими та магістерськими дисертаціями, а згодом і над дисертаціями на здобуття наукового ступеня доктора філософії. Дійсно редакційні колегії журналів заохочують розміщення коду у вільному доступі одночасно з публікацію наукової статті. Мова програмування python має всі пакети (бібліотеки), необхідні для здійснення складних наукових обчислень (як чисельних, так і аналітичних). Частину цих бібліотек мови python запозичено з відповідних бібліотек мови програмування Fortran та C, які є всесвітньовідомим надбанням.

Сучасний стан розвитку мов програмування та пакетів прикладних програм для наукових та інженерних обчислень характеризується конкуренцією між безкоштовними мовами (на кшталт мови програмування python) та платними програмами (Mathcad, Matlab, Wolfram Mathematica, Maple тощо). Платні програми часто мають більш простий синтаксис для чисельних та аналітичних розрахунків, але вужчий функціонал, ніж мова python. Надзвичайно широкий спектр методів та навіть в багатьох випадках їх надлишок (тобто одну задачу можна розв'язати декількома методами) – це особливість мови python, пов'язана з наявністю багаточисельної спільноти програмістів та користувачів, які її активно підтримують та розвивають. Таким чином, пишуться нові та підтримуються існуючі пакети, що обслуговують практично всі природничі науки від біотехнології (наприклад, пакет Biopython) до специфічних напрямків фізики твердого тіла (наприклад, ubermag – це пакет, який націлений на задачі магноніки), а також пакети для науковців та інженерів загального призначення (наприклад, пакети numpy, scikit-learn, scipy, sympy, pandas; py-pdf – це пакет для розв'язання диференціальних рівнянь з частинними похідними, в тому числі найбільш популярних рівнянь математичної фізики – рівняння дифузії, рівняння теплопровідності, рівняння Лапласа, хвильове рівняння, рівняння Шредингера).

Цей підручник охоплює в основному початковий рівень оволодіння інструментами мови програмування python для обчислень в науковій роботі біотехнолога, фізика, інженера. Матеріал даного підручника є основою для подальшого більш поглибленого вивчення пакетів numpy, pandas, scikit-learn, scipy, sympy мови програмування python для наукових та інженерних задач.

Підручник знайомить студентів з основами програмування на мові python та з низкою пакетів мови програмування python для аналізу експериментальних даних в біотехнології, фізиці та інших природничих науках: пакетом matplotlib для побудови графіків та діаграм, пакетами python для задач регресійного аналізу, пакетом opencv-python для аналізу зображень оптичної мікроскопії, скануючої зондової мікроскопії тощо. Підручник знайомить майбутніх біотехнологів та біофізиків з основами біоінформатики, яка є лідером в розробці лікарських препаратів, знайомить з пакетом Biopython мови програмування python та іншими програмними засобами, що використовуються для аналізу та порівняння генетичних даних, а також для розрахунку низки фізико-хімічних властивостей білків. Пакет Biopython мови програмування python є колекцією некомерційних інструментів для обчислень в біології, біофізиці та біоінформатиці з відкритим кодом. Пакет Biopython – це набір модулів python, які надають функції для роботи з послідовностями ДНК, РНК і білковими

послідовностями, забезпечуючи такий функціонал як зворотна комплементація ланцюжка ДНК, пошук мотивів в білкових послідовностях тощо. Він надає безліч синтаксичних аналізаторів для читання всіх основних генетичних баз даних, таких як GenBank, SwissPort, PDB, а також оболонки /інтерфейси для запуску інших популярних програм / інструментів біоінформатики, таких як NCBI BLASTN, Entrez тощо, в середовищі python.

Місце дисципліни «Інформаційні технології», яка викладається на основі даного підручника, в структурно-логічній схемі навчання забезпечується дисциплінами, такими як загальнотехнічна дисципліна «Вища математика», а також базовим рівнем володіння англійською мовою не нижче рівня А2. У структурно-логічній площині програми підготовки бакалаврів з біотехнології, а також фізики та астрономії дисципліна «Інформаційні технології» базується на попередньо вивчених дисциплінах в школі. Вивчення дисципліни впливає на подальший шлях здобувачів вищої освіти.

Основні задачі, які стоять перед студентом під час вивчення навчальної дисципліни «Інформаційні технології» полягають в набутті навичок написання коду на мові програмування python для статистичного аналізу та графічної візуалізації експериментальних даних, зчитування експериментальних даних та їх запис у файли різного формату (.txt, .csv, xlsx), аналізу зображень, отриманих методами оптичної, електронної, скануючої зондової мікроскопії (наприклад, виділення контурів об’єктів, підрахунок їх довжини, площі, аналіз кольорів тощо).

РОЗДІЛ 1. Інтерпретована, об’єктно-орієнтована мова програмування python

1.1. Історія мов програмування

Перші в історії комп'ютерні програми писали за допомогою машинних кодів, які задавали кожну операцію в комп'ютері (додавання, віднімання, множення тощо). Програмісти з застосуванням таблиць вбивали цей код на перфокарти, робота програміста була дуже трудомісткою і кропіткою, будь-яка помилка в коді призводила до необхідності заново переробляти перфокарти. Комп'ютери першого покоління, в 1920-х-1950-х роках, використовували перфокарти в якості основного носія при зберіганні і обробці даних. Потім, протягом 1970-х – початку 1980-х років, перфокарти використовувалися тільки для зберігання даних і поступово їх замінили магнітними стрічками.

Поява комп'ютерної програми-транслятора, що призначена для генерації машинного (двійкового) коду з символьного представлення цього коду на мові асемблера, була величезним проривом тому, що дозволила для написання програми замість паперових перфокарт з отворами, що кодують біти інформації (0 та 1), використовувати слова, схожі на англійську мову (наприклад: add або mov). Асемблер – компілятор, що перетворює текст з мови асемблера на машинну мову.

Мова асемблера специфічна для конкретної комп'ютерної архітектури. Мова асемблера – мова, яка близька до машинної мови, мова низького рівня на противагу мовам програмування високого рівня, що частково, чи повністю абстрагуються від деталей реалізації команд процесора. Чим нижчий рівень мови програмування, тим ближча специфіка роботи програми до самого процесора, для якого вона написана. Мови низького рівня складніші й потребують більш вузької спеціалізації програміста, оскільки програма, написана на асемблері для одного типу процесорів, не завжди є придатною для роботи з іншими процесорами. Тому в даний час в індустрії інформаційних технологій в основному використовуються мови програмування високого рівня. Тим не менше використання мови асемблера практично не має альтернативи при створенні: драйверів обладнання та машинозалежних підсистем ядра операційної системи (ОС); програми, які повинні зберігатися в постійному запам'ятовуючому пристрої – пам'яті, яка використовується для зберігання масиву незмінних даних; в пристроях з обмеженою продуктивністю («прошивок» комп'ютерів і різних електронних пристроїв) тощо. Тобто на сьогодні на мові асемблера кодують тільки фрагменти програм, тоді як інша частина програми розробляється на тій чи іншій мові високого рівня.

1.1.1. Старі, але живі, мови та мертві мови

В період з 1954 по 1957 рік групою програмістів під керівництвом Джона Бекуса в корпорації IBM створено першу мову програмування високого рівня Fortran [1, 2]. Назва Fortran є скороченням від FORmula TRANslator (перекладач формул). Fortran широко використовується в першу чергу для інженерних і наукових обчислень. Одна з переваг сучасної мови Fortran – велика кількість написаних на ній програм і бібліотек підпрограм [1]. Бібліотеки підпрограм – це набір заздалегідь складених для комп’ютерів частин програм (модулів), призначених для використання як цілих частин при складанні нових програм. Бібліотека може означати те саме, що пакет або модуль, або декілька модулів. Є велика кількість, написаних на мові Fortran (в більшій частині на старих версіях мови) різних математичних бібліотек для матричної алгебри і розв’язку систем лінійних рівнянь, бібліотеки для розв’язання диференціальних рівнянь, інтегральних рівнянь і їх систем, апроксимації функцій, спеціальних функцій, математичної статистики тощо [1].

Такі бібліотеки (пакети) поставляються, як правило, з компілятором. Низка таких пакетів створювалася протягом десятиліть, є надбанням людства і є популярною в науковому середовищі і донині, наприклад, Міжнародна бібліотека математики та статистики IMSL

(International Mathematics and Statistics Library) – комерційна колекція програмних бібліотек з чисельних методів, функціональність яких реалізується мовами програмування C [3], Java, C# [1], .NET Framework і Fortran [1]. Перша бібліотека IMSL на мові Fortran була випущена в 1970 році [1], в 1991 вийшла версія для мови програмування C, 2002 – для мови Java, а в 2004 році для мови C#. Мова C# була випущена в 2001 році [1, 2] разом з програмною платформою Microsoft – .NET Framework [2], яка підтримує кілька мов програмування. У 2009 році для мови python випущена бібліотека PyIMSL Studio, доступна як для комерційного, так і для некомерційного використання. .NET Framework – це великий набір написаних фрагментів коду (шаблонів), який програмісти можуть використовувати, щоб швидше писати програми. Шаблон проєктування або патерн (design pattern) в розробці програмного забезпечення – це повторювана архітектурна конструкція, що представляє собою рішення проблеми проєктування в рамках деякого часто виникаючого контексту. Кілька років тому Microsoft відкрила вихідний код .NET Framework, дозволивши всім бажаючим вносити свій вклад в розробку платформи. В результаті Microsoft стала найактивнішою організацією на GitHub, а 4 червня 2018 корпорація Microsoft купила GitHub за 7,5 млрд доларів (ТСН). GitHub – один з найбільших вебсервісів для спільної розробки програмного забезпечення (ПЗ).

Більшість таких бібліотек подібних до IMSL, як вже зазначалося, є фактично надбанням людства: вони доступні у вихідних кодах, добре задокументовані, налагоджені і досить ефективні. Сучасний Fortran досі живий, але вже досить сильно відрізняється від того, що було раніше [1]. Так, сучасний Fortran (Fortran 95 і Fortran 2003) набув рис, необхідних для ефективного програмування, для нових обчислювальних архітектур, дозволяє застосовувати сучасні технології програмування, зокрема, узагальнене і модульне програмування, об’єктно-орієнтоване програмування (ООП), зберігаючи при цьому сумісність з більш ранніми версіями [1]. Одна з головних концепцій розвитку сучасної мови Fortran – засоби підтримки паралельності і векторні операції [1]. Векторний процесор – це процесор, в якому операндами деяких команд можуть слугувати впорядковані масиви даних – вектори. Векторний процесор відрізняється від скалярних процесорів, які можуть працювати лише з одним оператором в одиницю часу. Абсолютна більшість процесорів є скалярними або близькими до них. Векторні процесори були розповсюджені в галузі наукових обчислень, де вони були основою більшості суперкомп'ютерів, починаючи з 1980-х і до 1990-х років. Але різке збільшення продуктивності і активна розробка нових процесорів призвели до того, що векторні процесори були витіснені зі сфери повсякденних процесорів. В більшості сучасних мікропроцесорів є векторні розширення, крім того сучасні відеокарти та фізичні прискорювачі можна розглядати як векторні співпроцесори. Таким чином стара мова програмування Fortran залишається живою і на сьогодні.

В 1959 році створено мову програмування COBOL (COmmon Business Oriented Language), яка представляє собою компільовану мову програмування високого рівня, що використовується в економічній галузі і для розв'язання бізнес-задач [1, 2, 4]. Синтаксис мови COBOL дозволяє писати програми, текст яких близький до англійської мови, тому до програм на мові COBOL немає потреби додавати коментарі, тобто мова COBOL є самодокументованою [1]. Мова програмування COBOL розвивалася дуже швидко, так вже в 1997 році провідна світова дослідницька і консалтингова компанія у сфері інформаційних технологій Gartner Group (США) звітувала, що 80% підприємств світу використовують програми на COBOL з більш ніж 200 мільярдами рядків написаного коду, і ще 5 мільярдів рядків коду пишеться щорічно [1]. COBOL дозволяє ефективно працювати з великими об'ємами даних, він насичений різноманітними можливостями пошуку даних, їх розподіленням та сортуванням; при цьому складність структур даних, які можливо описати засобами мови COBOL, практично не обмежена [1]. На COBOL розробляються великі та дуже великі проєкти, що складаються з мільйонів рядків коду [1]. І на сьогодні, значна частина банківських транзакцій здійснюється за допомогою програм, написаних на мові COBOL. Для мови програмування COBOL було затверджено низку стандартів: в 1968, 1974,

1985 і 2002 роках [1]. Останній стандарт додав в мову підтримку об'єктно-орієнтованої парадигми.

В 1968 році створена алгоритмічна мова ALGOL 68 (ALGOrithmic Language 1968) [3]. Особливості мови ALGOL [3] стали типовими для більшості імперативних мов, створених пізніше неї. Імперативне програмування – стиль написання вихідного коду програми, парадигма програмування, основні риси якої – використання іменованих змінних, оператора присвоювання, використання складних типів даних, підпрограм тощо. Саме в мові ALGOL з'явилося уявлення про програму не як про послідовність команд, а як про блокову структуру, що складається з чітко описаних та відокремлених одна від одної частин [3]. Основний блок програми на мові ALGOL – це сама головна програма, яка містить частину, укладену в блок, обмежений парою ключових слів begin і end, а також опис підпрограм [3]. При цьому в блоці програми можуть виділятися підблоки (підпрограми). Кожна підпрограма – це програма в мініатюрі, що має власні, описані усередині неї дані, певний інтерфейс у вигляді імені та списку формальних параметрів. В мові ALGOL були виділені структурні керуючі конструкції: розгалуження, цикли, які виконуються багаторазово; вкладені набори операторів обмежені тими ж ключовими словами begin і end, що дало можливість описувати логіку програми без використання безумовних переходів – сумнозвісного оператора goto, що провокує на створення заплутаних і погано структурованих програм [3]. Сучасним програмістам подібна структура програми здається очевидною, хоча в певних рисах застарілою і не завжди зручною (часто критикуються нескінченні begin і end в програмах на Pascal [3], який успадкував цю особливість саме від ALGOL [3]), але на момент появи мови програмування ALGOL це було значним кроком вперед [3]. Програми ставали регулярними, це давало можливість нарощувати їх за обсягом, зберігаючи їх доступними для огляду, зрозумілими, для аналізу і виправлення. Вкрай важливою властивістю мови ALGOL стала можливість організації рекурсивних процедур, до цього у лідерів ринку Fortran та COBOL рекурсія була прямо заборонена [1, 3]. ALGOL був популярним в основному в Європі, а мовою Fortran користувалися в основному в США. Але в обох цих мовах помітна тенденція до спрощення процесу програмування в порівнянні з Ассемблером.

В той же час почали з’являтися мови програмування, які є «живими» і широко використовуються і сьогодні в тому сенсі, що продовжують займати значну частину ринку програмування. Так, у 1963-1964 роках професорами Дартмутського коледжу (один з університетів Ліги плюща США) Джоном Кемені і Томасом Курцом було створено мову програмування високого рівня BASIC (від англ. basic – початковий, елементарний) з метою отримання простої в користуванні мови для початківців [2]. Проте мова BASIC набула поширення у 1980-х роках і лишається популярною й досі, маючи чимало діалектів. Ліга плюща – асоціація восьми найстаріших університетів США: Браунський університет, Дартмутський коледж, Гарвардський університет, Колумбійський університет, Корнельський університет, Єльський університет, Пенсільванський університет, Принстонський університет. Назва ліги походить від гілок плюща, що обвивають старі будівлі у цих університетах. Вважається, що члени ліги відзначаються високою якістю освіти.

В 1970 році Ніклаус Вірт, швейцарський програміст і теоретик програмування, професор разом з англійським колегою Чарльзом Ентоні Гоаром створив мову програмування Pascal [2, 3]. В 1970 роках Ніклаус Вірт розробив, разом з Чарльзом Гоаром та нідерландським ученим Эдсгером Вибе Дейкстра технологію структурного програмування. У 1971 році вийшла стаття Ніклауса Вірта «Розробка програми методом покрокового уточнення», в якій описано і обґрунтовано те, що стало згодом класичною методологію розробки програмного забезпечення «зверху вниз». Для перенесення Pascal-системи на різні обчислювальні платформи у 1973 році за участю Ніклауса Вірта був розроблений прототип віртуальної машини (абстрактної Pascal-машини), що виконує на будь-якій платформі проміжний P-код, в який передбачалося компілювати всі програми, тобто був розроблений компілятор ETH Pascal [3]. ETH Pascal став однією з перших реалізацій мов високого рівня, написаних на самій собі, на два роки випередивши компілятор мови C. Pascal належить до ALGOL-

подібних мов програмування, оскільки використовує семантику мови ALGOL [3]. Однак Pascal мав суттєве удосконалення – статичну типізацію [3]. Це означало, що присвоювання можна було виконувати лише для змінних, що належать до одного типу (одночасно вказувались правила, за якими типи вважались однаковими). Це удосконалення суттєво покращило стиль програмування, оскільки значну частину помилок вдавалось виявити ще на етапі компіляції, що збільшує надійність програм. Однак мова розроблялась як дослідницький проєкт, і первісний Pascal був мало придатний для написання великих проєктів, оскільки програму не можна було скласти з кількох програмних частин, просто не було передбачено такої можливості. Але ця мова програмування швидко завоювала популярність у навчальних закладах при вивченні програмування. А коли з'явились діалекти мови, де можливим стало окреме компілювання програмних частин, Pascal став засобом написання великих програмних систем [3]. Мова програмування Pascal стандартизована в багатьох країнах, а у 1983 році для Pascal було прийнято міжнародний стандарт (ISO 7185:1983).

У 1972 році Деннісом Рітчі у Bell Telephone Laboratories, США (з 1996 року – дослідницький підрозділ корпорації Lucent Technologies, США; з 2006 року дослідницький центр у корпорації Alcatel-Lucent, Франція) розроблено мову C, яка представляє собою універсальну, процедурну, імперативну мову програмування загального призначення [1, 2]. Мову C розроблено з метою написання нею операційної системи UNIX. Хоча C і було розроблено для написання системного програмного забезпечення, наразі вона досить часто використовується для написання прикладного програмного забезпечення. Мова C є найпопулярнішою у світі мовою програмування за кількістю вже написаного на ній програмного забезпечення, доступного під вільними ліцензіями коду та кількості програмістів, які її знають. Версії компіляторів для мови C існують для багатьох операційних систем та апаратних архітектур. Мова C здійснила великий вплив на інші мови програмування [1, 3], особливо на C++ [5], яку спочатку проєктували як розширення для C [3], а також на Java та C# [1], які запозичили у мови програмування C синтаксис [3].

1.1.2. Мови програмування родом з 80-х років

У 1980-1983 роках Б'ярном Страуструпом в Bell Telephone Laboratories, США розроблено мову програмування C++ (Сі-плюс-плюс) [2, 5]. C++ – це мова високого рівня з підтримкою кількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної [6]. Узагальнене програмування (англ. generic programming) – це парадигма програмування, що полягає в такому описі даних і алгоритмів, який можна застосовувати до різних типів даних, не змінюючи сам опис. У тому чи іншому вигляді підтримується різними мовами програмування. Можливості узагальненого програмування вперше з'явилися в 1970-х роках, в у мові програмування Ada (вихідний варіант, стандартизований 1983 р.) [2, 5], а потім у багатьох об'єктно-орієнтованих мовах, таких як C++ [5], Java [2] і мовах для платформи .NET. Термін «узагальнене програмування» вперше було введено Девідом Массером і Олександром Степановим, які описували парадигму програмування, засновану на тому, що типи даних і структури даних є абстрактними і не впливають на конкретну реалізацію алгоритмів, а загальні функції реалізовані з використанням узагальнених формалізованих типів. Так, наприклад, функція обміну значень двох змінних (об'єктів) може виконуватися незалежно від типу цих об'єктів.

Мова C++ початково отримала назву «C з класами». Згодом Б'ярн Страуструп перейменував мову на C++ у 1983р. [2, 5]. Мова програмування C++ вперше описана стандартом ISO/IEC 14882:1998. У 1990-х роках C++ стала однією з найуживаніших мов програмування загального призначення. Мову використовують для системного програмування, розробки програмного забезпечення, написання драйверів, потужних серверних та клієнтських програм, а також для розробки розважальних програм, наприклад,

відеоігор. Як вже зазначалося, C++ суттєво вплинула на інші популярні сьогодні мови програмування такі як C# [1] та Java.

1.1.3. Скриптові мови програмування з 90-х років

Але технічний прогрес не стояв на місці, і на межі 80-х і 90-х років комп'ютери стали мати настільки високу швидкодію та великий об’єм пам’яті, що для спрощення читання та відповідно внесення змін до кодів можна вже було робити і неефективні речі (з точки зору економії пам’яті і часу, необхідних для роботи програми). Зокрема, з’явилися так звані скриптові мови, що не компілювалися в машинний код, а інтерпретувалися, але ці скриптові мови були призначені в основному для обробки текстів – це, наприклад, такі мови програмування як perl [2], python (він був тоді не дуже відомий) [2, 7–9], PHP (Personal Home Page) [2], ruby [2], тобто ті скриптові мови, які є популярними і на сьогодні.

Інтерпретовані мови програмування мають такі переваги:

- Вбудовані складні типи даних і операції над ними.
- Зручна робота з рядками за допомогою регулярних виразів.
- Можливість використовувати спільно різні програми.
- Відносна простота вивчення і використання.

Одночасно скриптові мови мають такі недоліки:

- Більший час виконання скрипта.
- Результат виконання скрипта залежить від версії інтерпретатора.
- Проблеми безпеки (скриптові мови на web сторінках можуть використовувати вразливість браузера і операційної системи). У комп'ютерній безпеці, вразливість (англ. system vulnerability) – це нездатність системи протистояти реалізації певної загрози або сукупності загроз. Тобто, це певні недоліки в комп'ютерній системі, завдяки яким можна навмисно порушити її цілісність і викликати неправильну роботу. Вразливість може виникати в результаті допущених помилок програмування, недоліків, допущених при проєктуванні системи, ненадійних паролів, вірусів та інших шкідливих програм, скриптових і SQL-ін'єкцій (один з поширених способів злому сайтів та програм, що працюють з базами даних, заснований на впровадженні в запит довільного SQL-коду).

При цьому не існує чіткої межі між тим, що називають «скриптом», а що називають «програмою».

Особливості скриптових мов:

- Скрипт, як правило, виконується інтерпретатором, окрема стадія компіляції відсутня.
- Наявність складних типів даних (списки, словники, кортежі, множини (набори), хеш-таблиці тощо).
- Динамічна типізація. Тип змінної визначається в залежності від того, яким чином змінна використовується.
- Наявність підпрограми «прибиральника сміття».

Підпрограма «прибиральник сміття» (англ. garbage collection) – це одна з форм автоматичного керування оперативною пам'яттю комп'ютера під час виконання програм. Підпрограма «прибиральник сміття» вивільняє пам'ять від об'єктів, які не будуть використовуватись програмою в подальшому. Цю підпрограму було винайдено Джоном Мак-Карті в 1959 році для розробленої ним мови програмування LISP (LISt Processing, обробка списків) [2, 10].

Протягом 1990-х років Інтернет став дуже популярним засобом обміну інформацією, об'єднавши у собі більшість існуючих на той час мереж. Завдяки відсутності єдиного керівного центру, а також завдяки відкритості технічних стандартів Інтернету, що автоматично робило мережі незалежними від бізнесу чи уряду, об'єднання виглядало

неймовірно привабливим. До 1997 року в Інтернеті вже нараховувалось близько 10 мільйонів комп'ютерів і було зареєстровано понад мільйон доменних назв.

Для веброзробок використовуються мови програмування PHP (Hypertext Preprocessor, гіпертекстовий препроцесор), попередня назва: Personal Home Page Tools – скриптова мова програмування, яка була створена для генерації HTML- сторінок на стороні вебсервера. PHP є однією з найпоширеніших мов, що використовуються у сфері веброзробок разом із Java, .NET, JavaScript, python, Ruby [8]. PHP – проєкт відкритого програмного забезпечення, набув розвитку як об'єктно-орієнтована мова програмування. Мова програмування PHP підтримує всі три основні механізми ООП – інкапсуляцію, поліморфізм підтипів і спадкування (батьківський клас вказується за допомогою ключового слова extends після імені класу).

1.1.4. Компільовані мови програмування з 90-х років

В 90-ті роки мова C++ застосовувалася практично для всього, що потрібно було писати не для інтернету, не для обробки тексту, а просто для додатків, для операційних систем, для ігор тощо [5]. Але мова C++ має низку недоліків, тому що ця мова успадкувала всі проблеми і недоліки мови C [5]. Важливо те, що для програмування на C++ повинна бути дуже висока кваліфікація програміста, якої немає у широкого загалу інженерів та дослідників. Тому в 1995 році компанією «Sun Microsystems» випущена мова програмування Java [5]. Мова Java – це об'єктно-орієнтована мова програмування [5]. З 2009 року удосконаленням цієї мови займається компанія «Oracle», яка того року придбала компанію «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. Компанія «Oracle» надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License. Мова Java в значній мірі запозичила синтаксис з мови C [3], проте відбулася важлива модифікація, а саме: усунуто можливість появи низки конфліктних ситуацій, що могли виникнути через помилки програміста, тобто значно полегшено сам процес розробки об'єктно-орієнтованих програм [5]. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині [5]. Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з мовою C++ зменшує швидкість роботи програм [5]. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування. Тобто мова Java розроблялася, щоб знизити вимоги до кваліфікації програмістів, щоб більш широке коло програмістів мало можливість писати якісні програми на Java [5].

Проєкт з розроблення мови C# (вимовляється Сі-шарп) був початий в 1998 році [1]. Мова C# 1.0 остаточно вийшла у 2001 році разом з програмною платформою Microsoft – .NET Framework, яка підтримує кілька мов [1, 2]. Мова C# – це об'єктно-орієнтована мова програмування [1] з безпечною системою типізацією для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою компанії Microsoft Research, яка належить Microsoft. Синтаксис C# близький до C++ і Java [1]. Мова має сувору статичну типізацію [6], підтримує поліморфізм, що полягає в можливості одночасного існування в одній зоні видимості декількох різних варіантів застосування операторів, що мають одне й те саме ім'я, але різні типи аргументів, до яких вони застосовуються, використовує вказівники на функції – члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML.

.NET Framework – програмна технологія, запропонована Microsoft як платформа для створення як звичайних програм, так і вебзастосунків. Багато в чому є продовженням ідей та принципів, покладених в технологію Java. Однією з ідей програмної технології .NET є сумісність програм, написаних різними мовами. Хоча ця можливість рекламується Microsoft як перевага .NET, мова Java має таку саму можливість. Програмна технологія .NET – це великий набір написаних фрагментів коду (шаблонів), які програмісти можуть

використовувати, щоб швидше писати програми. Шаблон проєктування або патерн (design pattern) в розробці програмного забезпечення – повторювана архітектурна конструкція, що представляє собою рішення проблеми проєктування в рамках деякого часто виникаючого контексту.

1.1.5. Вибір мови програмування в залежності від задачі

Припустимо, перед вами стоїть завдання написати драйвер для відеокарти. Якою мовою ви будете користуватися? Мовою Java, Ruby або PHP?

Мова С – це одна з найкращих мов, яка відмінно підходить для таких завдань системного програмування, як написати операційну систему, написати драйвер тощо. Крім того, зараз з'являються різноманітні пристрої, які мають живлення від батарейки. Так як все більшого розвитку набуває інтернет речей, природно, що таких пристроїв для розвитку інтернету речей буде мільйони, вони повинні бути дуже дешевими і споживати дуже мало електроенергії. Відповідно, в таких пристроях буде приблизно 2 кбайта пам'яті, процесор на 5 кГц, внаслідок чого вбудувати в них більш енергоємну віртуальну машину або скриптову мову найближчим часом не вийде, таким чином, перспективним буде написання програм для керування такими пристроями на мові С.

1.1.5.1. Вебпрограмування

Припустимо, ви хочете написати новий Facebook (соціальну мережу). На чому ви будете писати такий код? Для програмування на стороні сервера і на стороні клієнта, як правило, використовуються різні мови програмування.

Програмування на стороні сервера – це написання коду, який виконується на сервері, з використанням мов таких як Java [5], Rust, Ruby, PHP, C# [1], python [7], які підтримуються сервером, також можливе написання коду, який виконується на стороні сервера на мові JavaScript (фреймворк Node.js) [8].

Сервер як комп’ютер – це комп’ютер у локальній чи глобальній мережі, який надає користувачам свої обчислювальні і дискові ресурси, а також доступ до встановлених сервісів, найчастіше працює цілодобово, чи у час роботи групи його користувачів.

Сервер як програма – це програма, яка надає деякі послуги іншим програмам (клієнтам). Зв'язок між клієнтом і сервером зазвичай здійснюється за допомогою передачі повідомлень, часто через мережу, і використовує певний протокол для кодування запитів клієнта і відповідей сервера.

Мережевий протокол – у комп’ютерних мережах – це набір правил, що визначені для комп’ютерів у мережі. Протокол також задає загальні правила взаємодії різноманітних програм, мережевих вузлів чи систем і створює таким чином єдиний простір передачі даних.

Веббраузери взаємодіють з вебсерверами за допомогою гіпертекстового транспортного протоколу HTTP (HyperText Transfer Protocol) – протокол передачі даних (гіпертекстових документів), що використовується в комп’ютерних мережах. Коли користувач натискає на посилання на вебсторінці, заповнює форму або запускає пошук, HTTP-запит відправляється з його браузера на цільовий сервер. Браузер (англ. browser) також переглядач, вебпереглядач, вебоглядач, вебнавігатор – програмне забезпечення для комп’ютера або іншого електронного пристрою, як правило, під'єднаного до Інтернету, що дає можливість користувачеві взаємодіяти з текстом, рисунками або іншою інформацією на гіпертекстовій вебсторінці. Тексти та рисунки можуть містити посилання на інші вебсторінки, розташовані на тому ж вебсайті або на інших вебсайтах. Браузер за допомогою гіперпосилань дозволяє користувачеві швидко та просто отримувати інформацію, розміщену на багатьох вебсторінках. Приклади найпопулярніших браузерів: Google Chrome, Safari, Edge, Mozilla, Firefox, Opera. Адресування сторінок відбувається за допомогою URL (Uniform Resource Locator, єдиний вказівник на ресурс), який інтерпретується, як адреса, що починається з http: для протоколу HTTP.

Існують так звані статичні і динамічні сайти. Схема нижче показує базову архітектуру вебсервера для статичного сайту. Статичний сайт (Рисунок 1.1) – це такий, що повертає один і той же жорстко закодований вміст з сервера щоразу, коли користувачем запитується конкретний ресурс. Коли користувач хоче перейти на сторінку, браузер відправляє HTTP-запит із зазначенням його URL. Сервер доставляє замовлений документ зі своєї файлової системи і повертає HTTP-відповідь, що містить документ і успішний статус (зазвичай 200 OK). Якщо файл не може бути вилучено з яких-небудь причин, повертається статус помилки (помилки клієнта або помилки сервера).

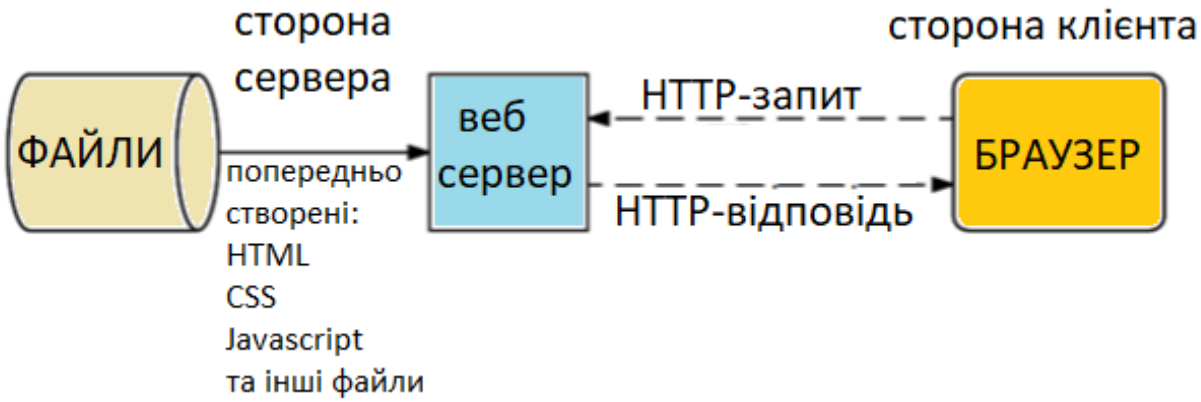


Рисунок 1.1 – Діаграма статичного сайту

Більшість великих вебсайтів використовують програмування серверної частини, щоб динамічно відображати різні дані при необхідності, в основному взяті з баз даних, розміщених на сервері, і відправляти ці дані клієнту для відображення через деякий код (наприклад, HTML і JavaScript).

Найзначніша сучасна можливість програмування серверної частини в тому, що воно дозволяє формувати контент вебсайту під конкретного користувача, тобто створювати динамічні сайти (Рисунок 1.2). На динамічному вебсайті HTML-сторінки зазвичай створюються шляхом вставки даних з бази даних в HTML-шаблони (це набагато більш ефективний спосіб зберігання великої кількості контенту, ніж використання статичних сайтів). Тобто динамічні сайти можуть виділяти контент, який більш актуальний в залежності від уподобань і звичок користувача, при цьому відповіді генеруються динамічно тільки при необхідності. Це дозволяє спростити використання сайтів за рахунок збереження особистих переваг і інформації повторного використання, наприклад, збережених даних кредитної картки для оптимізації подальших платежів тощо. Це також дає можливість взаємодіяти з користувачем сайту, посилаючи повідомлення і оновлення відповідних даних по електронній пошті або по іншим каналах. Всі ці можливості дозволяють глибше взаємодіяти з користувачами.

Велика частина коду для підтримки динамічного вебсайту повинна виконуватися на сервері. Створення цього коду відомо, як «програмування серверної частини» (або іноді «програмування бекенд»). Схема (Рисунок 1.2) показує просту архітектуру динамічного сайту. Як і на попередній схемі, браузери відправляють HTTP-запити (2) в код серверної частини (показано на діаграмі як вебзастосунок). Запити динамічних ресурсів обробляються так само, як і для статичних сайтів, статичні ресурси – це будь-які файли, які не змінюються, зазвичай написані на CSS (Cascading Style Sheets; мова, що використовується для оформлення HTML-документу, яка описує, як мають відображатися елементи HTML), JavaScript, зображення, попередньо створені PDF-файли тощо. Для динамічних запитів (Рисунок 1.2) сервер інтерпретує запит, читає необхідну інформацію з бази даних (3), комбінує отримані дані з шаблонами (фреймворками) HTML і повертає відповідь, що містить згенерований HTML-документ (5, 6). Веббраузери отримують HTML-документ із сервера по

протоколам HTTP або відкривають локальний диск, що дає змогу інтерпретувати код в інтерфейсі, який буде зареєстрований на екрані монітора.

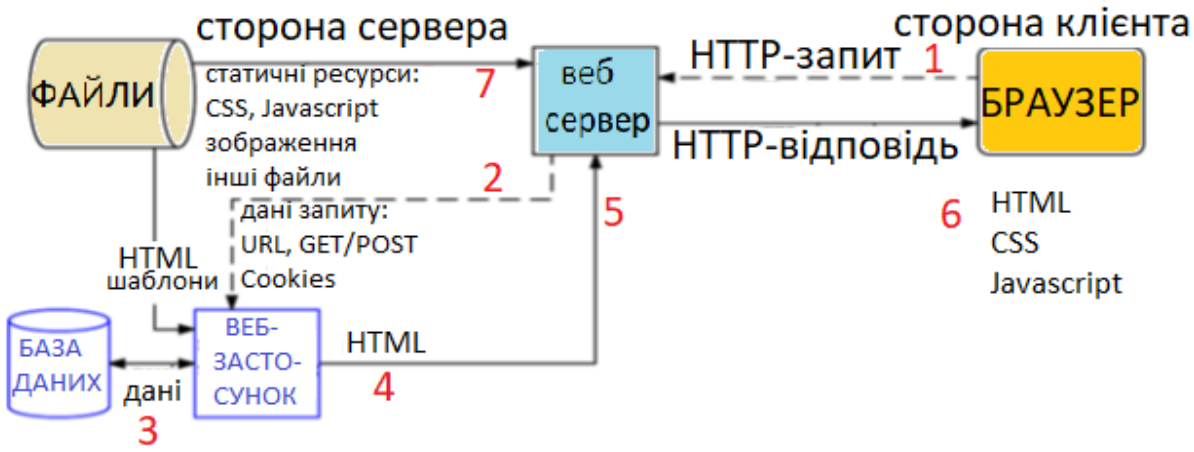


Рисунок 1.2 – Діаграма динамічного сайту

Методи GET і POST (Рисунок 1.2) використовуються для відправки даних на сервер. Частіше всього методи використовуються в HTML формах та гіперпосиланнях. POST і GET запити можна відправити на сервер за допомогою будь-якого програмного забезпечення, що працює з протоколом HTTP. Процедура розгляду запитів може відрізнятися в залежності від типу сервера. Більшість діючих сайтів працюють з мовою програмування PHP. У цьому випадку дані, що передаються, потрапляють в суперглобальні масиви $_GET і $_POST. Основна відмінність методу GET від POST в способі передачі даних. Запит GET передає дані в URL у вигляді пар "ім'я-значення" (іншими словами, через посилання), а запит POST передає дані в тілі запиту. Ця різниця визначає властивості методів і ситуації, які підходять для використання того чи іншого HTTP методу. Сторінка, створена методом GET, може бути відкрита повторно безліч разів, може бути проіндексована пошуковими системами і додана в закладки користувачем. З цього випливає, що метод GET слід використовувати для отримання даних від сервера і не бажано в запитах, які передбачають внесення змін до ресурс. Наприклад, можна використовувати метод GET в HTML формі фільтра товарів: коли потрібно, виходячи з даних введених користувачем, переправити його на сторінку з відфільтрованими товарами, відповідними його вибору. Запит, виконаний методом POST, навпаки слід використовувати у випадках, коли потрібно вносити зміни в ресурс (авторизувати, відправити форму оформлення замовлення, форму зворотного зв'язку, форму онлайн заявки). Повторний перехід по посиланню не приведе до повторної обробки запиту, тому що не буде містити переданих раніше параметрів. Метод POST має більшу ступінь захисту даних, ніж GET: параметри запиту не доступні для користувача без використання спеціального програмного забезпечення, що дає методу переваги при пересиланні, наприклад, конфіденційних даних.

Coockie (Рисунок 1.2) – невеликий фрагмент даних, відправлений вебсервером, який зберігається на комп'ютері користувача. Вебклієнт (веббраузер) при кожній спробі відкрити сторінку відповідного сайту пересилає цей фрагмент даних вебсервера в складі HTTP-запиту. Застосовується для збереження даних на стороні користувача.

Програмування на стороні клієнта (Рисунок 1.1, Рисунок 1.2) – це написання коду на мовах, таких як JavaScript, і цей код може виконуватися браузером.

Як вже зазначалося, коди клієнтської та серверної частини істотно відрізняються:

- Вони мають різні цілі і призначення.
- Як правило, вони не використовують одні і ті ж мови програмування (виняток становить JavaScript, який можна використовувати на стороні сервера і клієнта).

- Вони виконуються в різних середовищах операційної системи.
- Код, який виконується в браузері, відомий як код клієнтської частини, перш за все пов'язаний з інтерпретацією зовнішнього вигляду і поведінки відображання вебсторінки. Це включає в себе вибір і стилізацію компонентів для користувача інтерфейсу, створення макетів, навігацію, перевірку форм тощо. Навпаки, програмування вебсайту на стороні сервера в основному включає вибір вмісту, який повертається браузеру у відповідь на запити. Код на стороні сервера обробляє такі завдання, як перевірка відправлених даних і запитів, використання баз даних для зберігання та вилучення даних і відправка правильних даних клієнта в міру необхідності.
- Код клієнтської частини написаний з використанням HTML, CSS і JavaScript, він запускається в веббраузері і практично не має доступу до базової операційної системи (включаючи обмежений доступ до файлової системи). Веброзробники не можуть контролювати, який браузер може використовувати кожен користувач для перегляду вебсайту, браузери забезпечують різні рівні сумісності з функціями коду на стороні клієнта, і одним із завдань програмування на стороні клієнта є обробка відмінностей у підтримці браузера.
- Код серверної частини може бути написаний на будь-якій кількості мов програмування. Приклади популярних мов серверної частини включають в себе PHP, python [7], Ruby, C # [1] і JavaScript [8]. Код серверної частини має повний доступ до операційної системи сервера, і розробник може вибрати, яку мову програмування (і скільки разів) він хотів би її використовувати.
- Розробники зазвичай пишуть свій код, використовуючи шаблони, так звані вебфреймворки. Вебфреймворки – це набори функцій, об'єктів, правил та інших конструкцій коду, призначених для вирішення спільних проблем, прискорення розробки та спрощення різних типів завдань, що стоять в конкретній галузі. І знову, оскільки і клієнтська і серверна частини використовують фреймворки, області дуже різні і, отже, фреймворки теж різні. Фреймворки клієнтської частини спрощують верстку і уявлення даних, тоді як фреймворки серверної частини забезпечують функціональність вебсервера, яку, в іншому випадку, повинні були здійснювати користувачі самостійно (наприклад, підтримка сесій, підтримка користувачів і автентифікація, простий доступ до бази даних, шаблонів бібліотек тощо). Фреймворки клієнтської частини часто використовуються для прискорення написання коду клієнтської частини, але написання коду без використання фреймворка може бути більш швидким і ефективним, якщо, наприклад, потрібен невеликий простий вебсайт. І, навпаки, ви практично ніколи не наважитеся написати код серверної частини вебдодатку без фреймворку: здійснення життєво важливої функції, такої як HTTP сервер, дійсно складно зробити з нуля, скажімо, на python [7], але вебфреймворки для python, такі як Django, забезпечують це поряд з іншими корисними інструментами [8].

Програмування на стороні сервера (Рисунок 1.1, Рисунок 1.2) дуже корисне, оскільки дозволяє ефективно доставляти інформацію, підібрану для індивідуальних користувачів і, таким чином, створювати набагато кращий досвід використання. Компанії, такі як Amazon, використовують програмування серверної частини для побудови дослідницьких результатів для товарів, формування цільової пропозиції, заснованої на перевагах клієнта і попередніх покупках, спрощення замовлень тощо. Банки використовують програмування серверної частини, щоб зберігати облікову інформацію і дозволяти тільки авторизованим користувачам переглядати і здійснювати транзакції. Інші сервіси, такі як Facebook, Twitter, Instagram і Wikipedia використовують програмування на стороні сервера щоб виділяти, поширювати і контролювати доступ до контенту.

1.1.5.2. Програмування для бізнесу

Припустимо, Ви хочете написати яку-небудь програму для банку. На якій мові програмування Ви будете писати таку програму? На python, як правило не пишуться

програми для банківської справи і для бізнесу, тому що на python в процесі написання дуже складно виявити всі помилки. В мові python помилка може бути прихована і може виникнути у несподіваній ситуації. Проте на python програми пишуться дуже швидко, що дозволяє на сьогодні цій мові захопити ринок. А якщо потрібно написати програму для бізнесу чи банківських розрахунків, щоб забезпечити надійність, добре підходить мова програмування Java або .Net. Як вже зазначалося до сих пір, значна частина банківських транзакцій і на сьогодні здійснюється за допомогою програм, написаних на мові COBOL [1]. В результаті повторюваних обчислень з плаваючою точкою накопичуються значні помилки округлення.
Особливо критичні накопичення значних помилок округлення при роботі з фінансами. Так, щоб реалізувати грошові операції в банківських програмах на Java, програмісти пишуть окремі класи для підрахунку сум та мають проблеми з правилами округлення.
Мова COBOL позбавлена від таких помилок [1]:

- По-перше, десяткові обчислення в ньому ведуться з фіксованою точкою, а не з плаваючою, як у багатьох сучасних мовах. В результаті операції проходять швидше, ніж у випадку з плаваючою точкою: не потрібні окремий співпроцесор і складні правила округлення.
- По-друге, діапазон пам'яті для зберігання змінних різних типів у COBOL не залежить від компілятора або архітектури комп'ютера, де запускається код. Для порівняння: мова C ++ гарантує тільки мінімальний розмір блоку пам'яті для кожного типу.

Ці особливості роблять мову COBOL оптимальною для роботи з грошовими сумами: число цифр після коми однакове і заздалегідь відоме, операції з цілими числами в пам'яті дають швидкий передбачуваний результат, округленням чисел управляє програміст, а не реалізація арифметико-логічного пристрою в конкретній моделі процесора.

1.1.5.3. Використання відразу декількох мов для розробки

Багато проєктів не пишуться якоюсь однією мовою, тобто у них частина якась написана на одній мові, частина – на іншій, ще якась частина – на третій. Наприклад, якщо у вас якийсь вебдодаток, який обробляє величезні обсяги інформації, звернення до дисків, то напевно він написаний на мові C, щоб швидко здійснювати запис на диск тощо. Писати весь проєкт на C не варто. Може бути, там якась проміжна логіка, написана на Java, яка в свою чергу звертається до функцій мови C. Ну а та частина, на яку дивиться користувач, може бути написана на одній із скриптових мов, на тих, що безпосередньо виконуються браузером (наприклад, JavaScript). І все це разом успішно взаємодіє і працює.

У розробці якихось застосунків, навіть великих, існує така практика: на python пишуть прототип (як програма буде працювати), продумують архітектуру. Писати на ньому дійсно дуже швидко, тобто можна швидко створити прототип, експериментувати з ним і за необхідності (у разі виявлення недоліків архітектури) ще раз створити вже новий прототип з іншою архітектурою. Здавалося б, при цьому двічі виконується робота, два рази зробили роботу, від цього в два рази більше часу пішло (або, в півтора). Але це зовсім не так! Часто виявляється, що такий спосіб найкращій, тому що, якщо Ви напишете відразу на чомусь, наприклад на Java, а потім вирішите: «Ні, давайте здійснимо рефакторинг, міняємо архітектуру повністю тощо» - то витратите в 10 разів більше часу.

1.1.6. Умови успіху будь-якої мови програмування

Тепер з’ясуємо питання, чому деякі хороші на вигляд мови не вижили, ну або живуть в дуже обмеженому просторі. Одну з причин можна зрозуміти на прикладі мови Oberon. Ця мова була мінімалістичною, тобто, там було дуже мало команд. Така мова не мала значного розвитку.

Можна навести ще один приклад. Для американських військових було розроблено мову Ada [5], на якій і до сьогоднішнього часу пишуть програми, але тільки для військових. Чому

деякі мови на зразок python, яку спочатку не підтримувала ніяка компанія, захопили ринок [8]? Мова PHP теж захопила значну частину ринку. А мільярди доларів вкладених, наприклад, в мову Ada не призвели до її розповсюдження [5]. Це пов'язано з тим, що немає інфраструктури навколо таких мов. Тобто мова може бути досконалою, але поки немає документації, поки немає спільноти, яка користується цією мовою і найголовніше, поки немає великої кількості бібліотек, мова не захоплює ринок. Тобто, ви, наприклад, захотіли на мові Oberon написати сайт. При цьому ви не зможете підключити якісь спеціалізовані бібліотеки, тому що вони мові Oberon відсутні. Розповсюдженими і популярними є ті мови, які вміють користуватися бібліотеками, створеними для неї самої і для інших мов, наприклад, як це робиться на python в тих задачах, для яких на мові python код відсутній [8]. Наприклад, мова python вміє взаємодіяти з низкою стандартних алгоритмів написані на мові C, Fortran [3]. Мова Java також має Java Native Interface подібно до python. Тобто успішні мови можуть взаємодіяти з уже існуючими бібліотеками (в основному бібліотеками мови C) і за рахунок цього ефективно працюють [8]. Поступово такі мови нарощують свою власну інфраструктуру і живуть добре. Тому ідея, що не варто писати програму на тих мовах, які не вміють підключати бібліотеки мови C, є зрозумілою.

Щоб мова була популярною і захопила ринок, необхідно:

- Існування спільноти, яка користується цією мовою
- Існування великої кількості бібліотек
- Можливість взаємодії з бібліотеками, створеними для неї самої і для інших мов, наприклад, як це робиться на мові python [7], для якої низка стандартних алгоритмів написана на мові C, Fortran [3].
- Наявність відкритого у доступі програмного забезпечення, можливість читати його вихідні тексти, вносити зміни, використовувати його частини в програмах. В основі вільного програмного забезпечення лежить ідея спільноти, яка ділиться своїми знаннями.

Мова python розроблена в 1990 році Гвідо ван Россум, він назвав цю мову на честь телешоу на BBC під назвою «Monty Python's Flying Circus») [7]. Python – це інтерпретована мова програмування, вихідний код якої, частинами перетворюється в машинний у процесі виконання спеціальної програми – інтерпретатора [7].

Python – це приклад відкритого у доступі програмного забезпечення – FLOSS (Free/Libré and Open Source Software), тобто можна вільно поширювати копії цього програмного забезпечення, читати його вихідні тексти, вносити зміни, використовувати його частини в своїх програмах [7]. В основі вільного ПЗ лежить ідея спільноти, яка ділиться своїми знаннями, тому python постійно поліпшується співтовариством і в цьому секрет постійного росту популярності, починаючи з 2014 року, на даний час python входить в трійку найпопулярніших мов програмування. Завдяки своїй відкритій природі, python був імпортований на багато платформ (тобто змінений таким чином, щоб працювати на них). Всі програми python можуть запускатися на різних платформах без будь яких змін, якщо тільки уникати використання системно-залежних функцій. Python можна використовувати в GNU/Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE тощо.

1.1.7. Відмінність між об'єктно-орієнтованим і системним програмуванням

Об'єктно-орієнтований підхід (ООП) був винайдений Крістеном Нюгордом і Оле-Йоханом Далем при розробці мови Simula 67 [2]. Згодом багато концепцій ООП було розвинено Аланом Кейем з колегами при роботі над мовою Smalltalk. Об'єктно-орієнтований підхід передбачає, що будь-яку програму можна представити у вигляді набору взаємодіючих об'єктів. Спочатку передбачалося, що об'єкти будуть взаємодіяти за допомогою обміну повідомленнями. Однак в більшості існуючих мов програмування об'єкти взаємодіють за

допомогою виклику функцій. Для опису об'єктів в мовах програмування існують класи. Фактично, клас – це модуль (шаблон), для якого можна створити екземпляр (або кілька примірників). Як і будь-який модуль, клас має свої інтерфейс (відкриту частину) і реалізацію (закриту частину). Примірник класу є об'єктом. Об'єктно-орієнтований підхід вважають розвитком структурного підходу до програмування. Основна відмінність полягає в тому, що об'єктно-орієнтований підхід дозволяє об'єднати дані і методи, які обробляють ці дані, в єдиній сутності – в об'єкті.

Системний підхід – це підхід до:

- вивчення об'єктів і явищ;
- побудови наукових теорій і гіпотез;
- проєктування техніки.

Основоположником системного підходу є австрійський біолог Карл Людвіг фон Берталанфі. Суть системного підходу полягає в тому, що будь-який об'єкт або явище розглядається, як система. Що це означає? Будь-яка система має призначення, мету, заради якої існує. Система складається з низки систем більш низького рівня, які називаються підсистемами. Підсистеми, впорядковані всередині системи, і утворюють її структуру. Система має системний ефект. Системний ефект – це те, що відрізняє систему від просто суми її частин. Системний ефект – це властивість системи, якою вона володіє, але якої не мають її частини. Наприклад, літак може літати, а його окремі частини – крила, фюзеляж, шасі літати не можуть. Системний ефект відповідає призначенню системи, і він проявляється завдяки правильній організації частин системи, тобто завдяки ретельно пропрацьованій структурі. З точки зору системи її підсистеми являють собою "чорні ящики". Система знає про призначення її окремих підсистем, але не знає, як вони влаштовані всередині, не знає їх структуру.

Системи утворюють ієрархію рівнів:

- Над-над-система.
- Над-система.
- Система.
- Під-система.
- Під-під-система.

Розглядаючи об'єкт або явище системно, можна переміщатися з одного системного рівня на інший. Якщо необхідно зрозуміти призначення системи, то потрібно піднятися на рівень вище і подивитися на систему з точки зору надсистеми: де система розташована, як і для чого вона використовується? Тобто призначення системи завжди лежить за її межами.

Якщо необхідно зрозуміти, як функціонує система, то потрібно спуститися на рівень нижче, тобто від надсистеми до системи, від призначення системи до її структури. Хороший інженер (конструктор або винахідник) ніколи не розглядає систему ізольовано – він завжди вивчає систему з різних системних рівнів: надсистемного, системного, підсистемного (Альтшуллер Г.С. "Структура талановитого мислення").

Сучасні програми – це складні технічні системи, які можна представити у вигляді ієрархії системних рівнів:

- Код.
- Функція.
- Об'єкт, клас (група даних і функцій).
- Група взаємодіючих об'єктів, класів.
- Функціональний модуль.
- Бібліотека.
- Набір бібліотек.
- Програма.
- Додаток (набір програм і ресурсів) тощо.

В чому різниця між об’єктно-орієнтованим і процедурним підходами до програмування? ООП – це такий підхід до програмування, де на першому місці стоять

об'єкти. Найчастіше під звичайним програмуванням розуміють процедурне програмування, в основі якого – процедури і функції. Функція – це міні-програма, яка отримує на вхід дані, виконує всередині себе якісь дії (розрахунки, друк, побудову графіків тощо) і може передавати отримані дані.

Наприклад, в інтернет-магазині, бібліотеці, лікарні тощо може бути функція «перевірити email». Вона отримує на вхід якийсь текст, зіставляє зі своїми правилами і видає відповідь: це правильна електронна адреса чи ні. Якщо правильна, то true, якщо ні – то false (Рисунок 1.3).

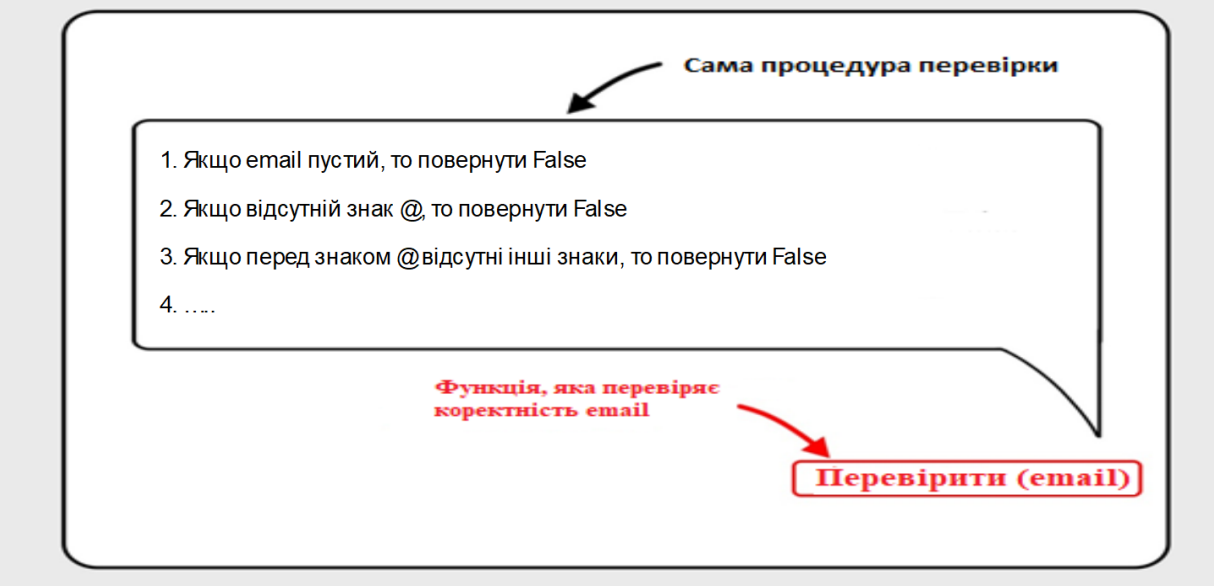


Рисунок 1.3 – Схема функції «перевірити email»

Функції корисні, коли потрібно упакувати багато команд в одну. Наприклад, перевірка електронної адреси може складатися з однієї перевірки на регулярні вирази, а може містити безліч команд: запити в словниках, перевірку по базах спамерів, зіставлення з уже відомими електронними адресами тощо (спамер – це людина або організація, що розповсюджують спам). У функцію можна упакувати будь-яку послідовність дій і потім просто викликати її однією командою.

Що не так з процедурним програмуванням? Процедурне програмування ідеально працює в простих програмах, де всі завдання можна вирішити десятком функцій. Функції акуратно вкладені одна в одну, взаємодіють одна з одною, можна передати дані з однієї функції в іншу.

Наприклад, Ви пишете функцію «Зареєструвати користувача бібліотеки». Всередині цієї функції вам необхідно перевірити електронну адресу користувача. Ви викликаєте функцію «перевірити email» всередині функції «зареєструвати користувача», і в залежності від відповіді функції Ви або реєструєте користувача, або виводите повідомлення про помилку. І у вас ця функція «перевірити email» зустрічається ще в десяти місцях. Уявіть, що прийшло розпорядження від менеджера ПЗ, щоб функція «зареєструвати користувача», повертала не true або false, а код помилки, щоб точно визначати, в чому помилка при введенні електронної адреси, наприклад, якщо email пустий, то код 01, якщо відсутній знак @ – код 02, якщо перед знаком @ відсутні інші знаки – код 03 і так далі (Рисунок 1.4).

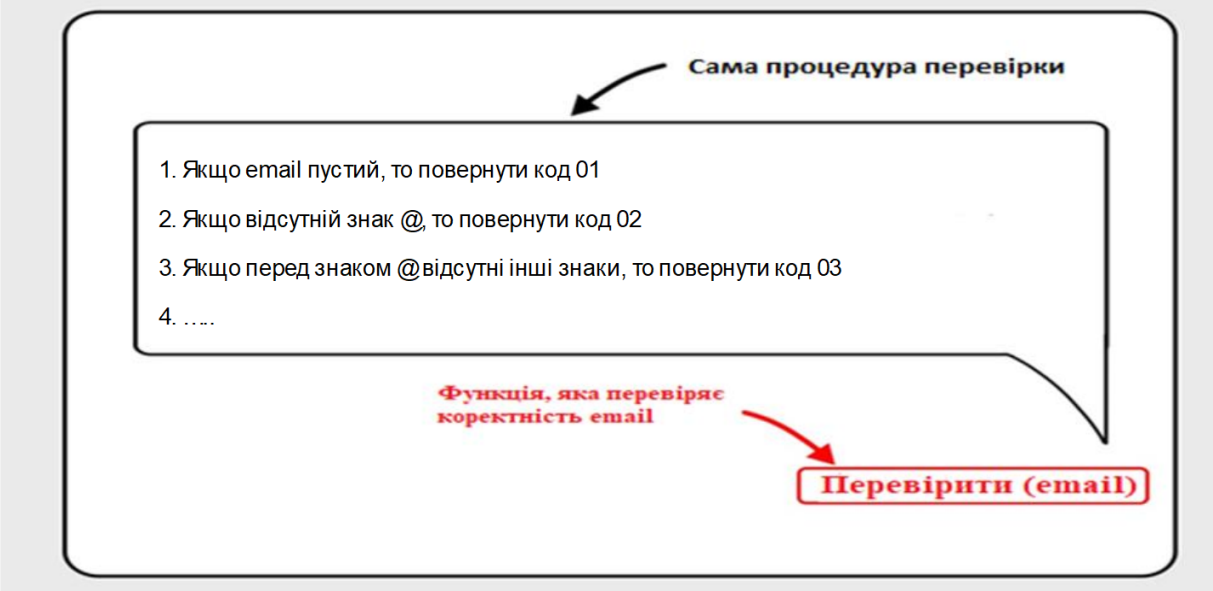


Рисунок 1.4 – Схема зміненої функції «перевірити email»

Такі зміни (Рисунок 1.4) реалізувати нескладно, тобто необхідно змінити функцію так, щоб вона замість true - false повертала код помилки, а якщо помилки немає – повертала «ОК». Але в результаті всі рядки в програмі, які викликали функцію «перевірити email» і очікували від перевірки true або false, тепер необхідно також змінити (Рисунок 1.5).

Тобто в програмі необхідно зробити зміни:

- або переписувати всі функції, щоб навчити їх розуміти нові відповіді перевірки email;
- або переробити саму перевірку адреси, щоб вона залишилася сумісною зі старими викликами, але в потрібному вам місці видавала коди помилок;
- або написати нову перевірку, яка видає коди помилок, а в старих місцях виклику функції «перевірити email», використовувати стару перевірку.

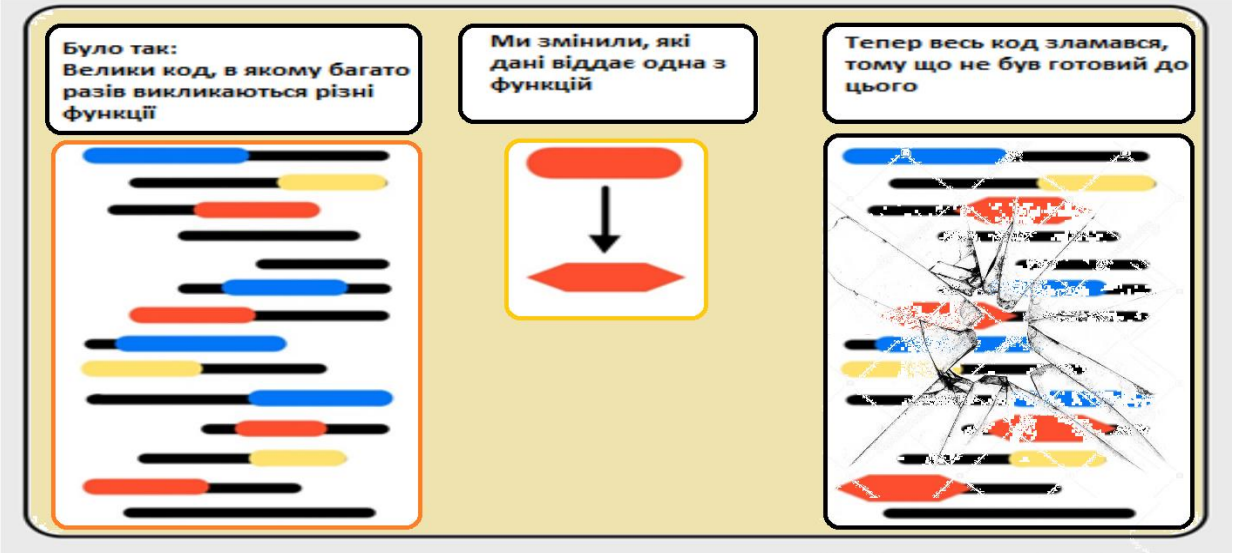


Рисунок 1.5 – Схема програми, яка використовує функцію «перевірити email»

Завдання можна вирішити за декілька годин, але уявіть, що у вас цих функцій не десяток, а сотні. І змін до них потрібно робити десятки в день. І кожна зміна, як правило, робить функції більш складними, і такі функції можуть повертати більш складний результат. При цьому кожна зміна в одному місці програми приводить до зміни в декількох інших місцях. В результаті будуть народжуватися десятки клонованих функцій, в яких Ви спочатку

будете розбиратися, але в кінці кінців це буле складно і навіть практично неможливо. Це на жаргоні програмістів називається спагеті-код, і для боротьби з ним якраз придумали об'єктно-орієнтоване програмування (ООП).

Основне завдання ООП – зробити складний код простіше. Для цього програму розбивають на незалежні блоки, які називаються об'єктами, дані всередині об'єкта будуть називатися властивостями, а функції – методами. Об'єкт – це набір даних і функцій – таких же, як в традиційному функціональному програмуванні. Можна уявити, що просто взяли частину програми і поклали її в коробку і закрили кришку. Ось ця коробка з кришкою є об'єктом. Наприклад, об'єктами можуть бути, термостат в лабораторії, студент КПІ, пацієнт в лікарні, мікроорганізм тощо.

У об'єкта є «інтерфейс»: у об'єкта є методи і властивості, до яких ми можемо звернутися ззовні цього об'єкта. Так само, як ми можемо натиснути кнопку на термостаті в лабораторії. У термостата є багато всього всередині, що змушує його працювати, але на головній панелі є тільки кнопки управління. Ось ці кнопки і є абстрактний інтерфейс. А все, що в середині об'єкта в ООП описується за допомогою шаблона об'єкта. В даному розділі вже обговорювалося створення шаблонів (фреймворків) при програмуванні динамічних сайтів.

Шаблон об'єкта в ООП – це клас. Наприклад, у вас може бути шаблон об'єкта – «студент» (його можна ще уявити як ідеальний об'єкт): в ньому Ви прописуєте все, що може характеризувати студента та відбуватися зі студентом. У шаблоні «студент» можуть бути властивості: ім'я, вік, адреса, копії документів, номер залікової книжки, номер картки в лікарні, участь в громадській, науковій, культурній спортивній діяльності університету тощо. І можуть бути методи: «призначити стипендію», «провести опитування», «оформити додаткову відомість на перездачу», «вручити відзнаку», «відрахувати» тощо. На основі цього шаблону об'єкту (тобто ідеального студента) Ви можете створити реального «Студента КПІ Петра Петренко». У нього при створенні будуть якісь властивості і методи, які Ви задали у ідеального студента, плюс можуть бути якісь свої, наприклад, «додати бали за участь у стартап проєкті». Шаблони об’єктів (їх можна уявити як ідеальні об'єкти), як вже відмічалося, називають класами. Такий підхід дозволяє програмувати кожен модуль незалежно від інших. Головне – заздалегідь продумати, як модулі будуть пов’язані один з одним і за якими правилами. При такому підході можна поліпшити роботу одного модуля, не зачіпаючи інші – для всієї програми неважливо, що всередині кожного блоку, якщо правила роботи з ним залишилися незмінними.

Розглянемо принципи ООП, а саме: інкапсуляція, успадкування, поліморфізм.

Інкапсуляція (англ. encapsulation) – це приховування деталей реалізації коду, яке дозволяє вносити зміни в частини програми безболісно для інших її частин, що істотно спрощує супровід і модифікацію програмних засобів.

Успадкування - створення нового класу об'єктів шляхом додавання нових елементів (методів) до вже наявного. Деякі об’єктно орієнтовані мови програмування дозволяють виконувати множинне успадкування, тобто об'єднувати в одному класі можливості кількох інших класів (Рисунок 1.6).

Поліморфізм – принцип ООП, який дозволяє за одним іменем метода закріпити декілька різних алгоритмів та реалізацій. Наприклад, метод, який називається volume, може мати декілька реалізацій, наприклад, здійснювати розрахунок об'єму (мікроорганізму, об’єкту) у формі сфери, циліндра, еліпсоїда тощо (Рисунок 1.7).

Розглянемо характерні для об’єктно орієнтованого програмування риси:

- Все в ООП є об’єктами.
- Всі дії та розрахунки виконуються шляхом взаємодії (обміну даними) між об’єктами, при якій один об’єкт потребує, щоб інший об’єкт виконав деяку дію. Об’єкти взаємодіють, надсилаючи і отримуючи повідомлення. Повідомлення – це запит на виконання дії, доповнений набором аргументів, які можуть знадобитися при виконанні дії.

- Кожен об’єкт має незалежну пам’ять, яка може складатися з інших об’єктів.
- Об’єкт може бути представником (екземпляром, примірником) класу, який виражає загальні властивості об’єктів. Клас – це шаблон, на основі якого може бути створений конкретний програмний об'єкт, який описує властивості і методи, що визначають поведінку об'єктів цього класу.

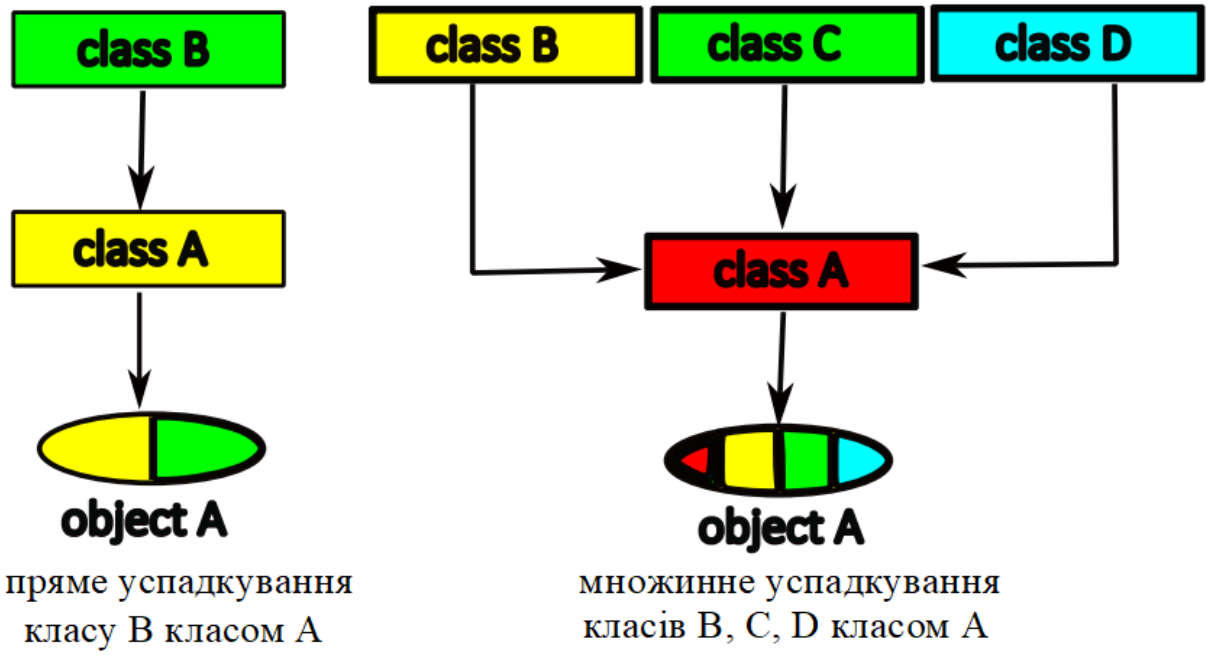


Рисунок 1.6 – Схема принципу ООП – прямого та множинного успадкування

У об'єктно-орієнтованого програмування багато плюсів, і саме тому цей підхід використовує більшість сучасних програмістів, тож відзначимо переваги ООП:

1. Візуально код стає простіше, і його легше читати. Коли весь код розбивається на об'єкти і у них є зрозумілий набір правил, можна відразу зрозуміти, за що відповідає кожен об'єкт і з чого він складається.
2. Менше однакового коду. Якщо в звичайному програмуванні одна функція підраховує повторювані символи в одновимірному масиві, а інша - в двовимірному, то у них більша частина коду буде однаковою. В ООП це вирішується спадкуванням.
3. Складні програми пишуться простіше. Кожну велику програму можна розкласти на кілька блоків, зробити їм мінімальне наповнення, а потім детально наповнити кожен блок.
4. Збільшується швидкість написання. На старті можна швидко створити потрібні компоненти всередині програми, щоб отримати мінімально працюючий прототип.

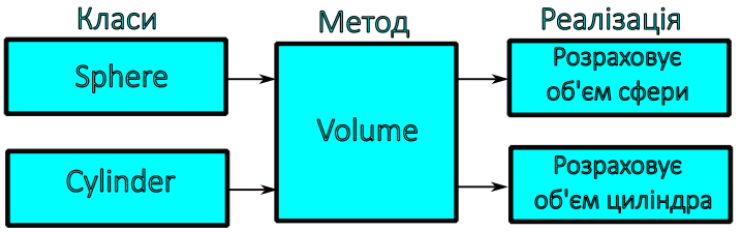


Рисунок 1.7 – Схема принципу ООП – поліморфізму

Недоліки ООП:

1. Складно зрозуміти і почати працювати. Підхід ООП набагато складніший, аніж звичайний підхід процедурного програмування – потрібно знати багато теорії, перш ніж буде написано хоч один рядок коду.

2. Вимагає більше пам'яті. Об'єкти в ООП складаються з даних, інтерфейсів, методів і багато чого іншого, а це займає набагато більше пам'яті, ніж проста змінна.
3. Іноді продуктивність коду буде нижче. Через особливості підходу частина задач може бути реалізована складніше, ніж могла б бути. Тому буває таке, що ООП-програма працює повільніше, ніж процедурна (хоча з сучасними потужностями процесорів це мало кого хвилює).
4. При використанні складної архітектури програми можуть створюватися проблеми із читанням коду, оскільки неправильне використання успадкування може приховувати методи у батьківських класах. Проте для малих проєктів та при правильній побудові архітектури проблеми з читанням коду через ООП не виникатиме.

Запитання до підрозділу «Історія мов програмування»

1. Що означає термін байткод?
2. Наведіть переваги мови програмування COBOL в економічній галузі і для розв'язання бізнес-задач.
3. Яка мова є найпопулярнішою у світі мовою програмування за кількістю вже написаного нею програмного забезпечення, доступного під вільними ліцензіями коду та кількості програмістів, які її знають?
4. Для яких застосунків використовується мова програмування C++ ?
5. Дайте визначення поняттю інтерпретатор.
6. Наведіть особливості скриптових мов.
7. Які умови успіху будь-якої мови програмування?
8. Перелічте принципи об’єктно-орієнтованого програмування (ООП).
9. Надайте визначення терміну інкапсуляція.
10. Які переваги об’єктно-орієнтованого програмування (ООП) Ви знаєте?
11. Які недоліки об’єктно-орієнтованого програмування (ООП)?
12. Дайте визначення терміну клас.
13. Для яких застосунків використовується мова програмування python?

1.2. Базові поняття мови python: базовий синтаксис, типи даних, змінні, введення та виведення даних, математичні операції, логічні оператори, умовні оператори, цикли

1.2.1. Типізація

Ключовим поняттям будь-якої мови програмування є типізація даних. Типізація визначає, які типи даних можуть бути використані в програмі та як вони взаємодіють.

Типізацію розділяють за двома категоріями: строга чи слабка, статична чи динамічна:

- Строга типізація (strong typing) – вимагає строгого дотримання типів при взаємодії з даними. Наприклад, неможливо додати ціле число до рядка без явного перетворення типів.
- Слабка типізація (weak typing) – дозволяє автоматичне перетворення типів у певних випадках. Це може призвести до неочікуваної поведінки програми.
- Статична типізація (static typing) – типи даних визначаються на етапі компіляції, і помилки, пов'язані з типами, виявляються під час компіляції. Це допомагає виявити більшу частину помилок на ранніх етапах розробки.
- Динамічна типізація (dynamic typing) – типи визначаються під час виконання програми. Це може дозволити більшу гнучкість, але може також призвести до помилок, якщо типи некоректно використані.

  Типізацію даних у python за наведеним описом можна назвати строгою динамічною [6]. Далі у розділах будуть наведені розширені пояснення цієї характеристики.

1.2.2. Типи даних

Тип даних – це категорія для значень операндів (змінних), і кожне значення належить саме одному типу даних. Також тип даних – це множина значень і операцій над цими значеннями. У python типи даних можна розділити на вбудовані в інтерпретатор (built-in) і невбудовані, які можна використовувати при імпортуванні відповідних модулів.

До найпростіших вбудованих типів відносять [11, 12]:

1. None Type – тип, який може мати лише значення None, що позначає невизначеність значення змінної.

Нижче наведено код, в якому використовується функція print. Ця функція виводить вказаний об’єкт на стандартний пристрій виводу (екран) або у файл текстового потоку. Наприклад, print('a') виводить літеру a. В нижченаведеному коді змінній result присвоюється print('a'). В наступному рядку коду окрім функції print використовується також функція type. Функція type повертає тип об'єкта в python. Таким чином, наступний рядок коду виводить тип змінної result. Як видно з результату роботи коду, цей тип – це None Type.

```
result = print('a')
print(type(result))

Результат роботи коду:
a
<class 'NoneType'>
```

2. Boolean Type – тип, який може мати два значення: True (правда), False (неправда).

```
print(5 < 11)
print(25 < (3 * 5))

Результат роботи коду:
True
False
```

3. Numeric Types – це група типів, які описують чисельні значення.

3.1. int – тип цілих чисел. Це значення може бути абсолютно будь-яким цілим числом. Наприклад, 0, 15, -67, 15000000. Для зручності можна використовувати символ "_", яким можна візуально розділити число на тисячі, мільйони, мільярди і далі, але при цьому саме число буде коректно інтерпретовано.

```
print(type(100_000))
print(type(15 * 10**25))
print(type(99999 * 10**999999))

Результат роботи коду:
<class 'int'>
<class 'int'>
<class 'int'>
```

3.2. float – тип чисел із плаваючою точкою (десятковий дріб). Це дійсні числа, які мають обмеження по значенням. Наприклад, найбільшим значенням є 1.7976931348623157e+308 (при збільшенні числа виникатиме помилка), а найменшим ненульовим – 4.94065645841246544e-324 (при зменшенні числа воно буде прирівняне нулю). Числа типу float не застосовують у банківській чи науковій сфері, де потрібні точні розрахунки, оскільки операції з ними можуть містити незначні помилки, які стають вагомими при виконанні складних розрахунків. Аналогічно до типу int, можна використовувати символ "_" для візуального розділення числа.

```
try:
   print(99999.9 * 10**9999)
except OverflowError:
    print('More then max float number')
print(17_976.9 * 10**304)
print(3.14159265359)
Результат роботи коду:
More then max float number
1.79769e+308
3.14159265359
```

3.3. complex – тип чисел, що відповідає математичним комплексним числам у вигляді (a + bj). Значення дійсної та уявної частини можуть бути як цілими, так і дійсними. Наприклад, 3 + 15j чи 7.16 + 44.2j.

```
a = 5 + 7j
print(type(a))
b = 15
print(type(b))
c = 3.5 + 7.8j
print(type(c))
print(a * b)
print(a * c)

Результат роботи коду:
<class 'complex'>
<class 'int'>
<class 'complex'>
(75+105j)
(-37.1+63.5j)
```

4. String Type (str) – тип, який відповідає послідовності символів, тобто тексту. Текст може бути як англійською мовою, так і будь-якою іншою мовою, що підтримується стандартом Unicode. Рядок створюється з використанням одинарних лапок, подвійних лапок та потрійних лапок. За допомогою потрійних лапок (" " " або ' ' ') можна створювати документацію до програм написаних на python. Наприклад, 'Іван', "ФБТ", ' ' 'Огляд можливостей python 3.11''', " " "Функція, яка виконує множення матриць" " ".

Як зазначалося в попередній лекції для мови python характерні такі складні структури даних:

- Рядок – для зберігання послідовності символів, він є незмінним.
- Список – для зберігання колекції різних типів даних, він є змінним.
- Кортеж – для зберігання колекції різних типів даних, він є незмінним.
- Набір – для зберігання колекції різних типів даних; він є змінним і зберігає тільки унікальні елементи.
- Словник – для зберігання колекції різних типів даних у вигляді пар: {ключове слово : значення}; для кожного значення ключове слово є унікальним.

1.2.3. Змінні

Змінна – це зарезервоване місце пам’яті для зберігання значень. Змінна ініціалізується (або створюється) перший раз, коли в ній зберігається значення. Після цього Ви можете використовувати змінну у виразах з іншими змінними та значеннями. Коли змінній присвоюється нове значення, старе значення забувається, це називається перезаписом

змінної. У мові python не потрібно оголошувати тип змінної вручну (як, наприклад, в C++, Pascal). Оголошення відбувається автоматично, коли Ви привласнюєте значення змінній, тобто тип змінної визначається під час операції присвоювання, це називається динамічна типізація.

Розглянемо на прикладі створення змінної: a = 10

При ініціалізації змінної на рівні інтерпретатора здійснюються такі кроки:

- створюється об'єкт із типом int та значенням 10 – в цей момент виділяється комірка пам'яті певного об'єму і в неї записуються дані про об'єкт;
- за оператором присвоєння '=' відбувається зв'язування змінної із назвою a та об'єкту типу int та значенням 10 (визначається адреса створеного об'єкту та створюється посилання на цей об'єкт через назву змінної).

Якщо після оператора зв'язування записано не об'єкт, а складний вираз, то перед створенням змінної виникає етап обчислення виразу до одного значення (див. розділ 1.2.5).

```
a = 10  # Створення змінної
print(a)  # Друк об'єкта через змінну
print(id(a))  # Друк ідентифікатора об'єкта
print(type(a))  # Друк типу даних об'єкта

Результат роботи коду:
10
140716654519368
<class 'int'>
```

Ідентифікатор (назва) змінних в python – це ім'я, яке використовується для ідентифікації змінної, функції, класу, модуля або іншого об'єкту python.

Ідентифікатор може містити тільки такі символи:

- літери в нижньому регістрі (від "a" до "z");
- літери у верхньому регістрі (від "A" до "Z") (python є регістр-чутливою мовою);
- цифри (від 0 до 9);
- нижнє підкреслення (_).

Ідентифікатор (назва) змінних не може співпадати з зарезервованими словами: import, as, false, true, and, or, not, in, if, elif, else, break, class, finally, is, return, none, continue, for, lambda, try, def, from, nonlocal, while, del, global, with, yield, assert, pass, except, raise.

Ідентифікатор (назва) не може починатися з цифри. Назви в python чутливі до регістру, тобто, PAM, pam, pAM та paM – це чотири різні змінні, назва змінної PAM є не коректною, при її використанні повідомлення про помилку при виконанні програми не буде, це умова правильного стилю написання коду на мові python – починати назви змінних з малої літери. Як правило з великої літери в python позначаються назви констант та класів. Приклади коректних імен: a; a1; a_b_95; _abc; _a; такі імена є некоректними: 1; 2a; 10_; A1; Sx5.

1.2.4. Введення та виведення даних

Введення даних з клавіатури в програму (починаючи з версії python 3.0) здійснюється за допомогою функції input(). Якщо ця функція виконується, то потік виконання програми зупиняється в очікуванні даних, які користувач повинен ввести за допомогою клавіатури. Після введення даних і натискання Enter, функція input() завершує своє виконання і повертає результат, який є рядком символів, введених користувачем. Дані повертаються у вигляді рядка, навіть якщо було введено число, наприклад:

```
b = input("Number: ")
print(type(b))
```

```
Результат роботи коду:
Number: 999
<class 'str'>
```

У змінній b зберігається не ціле число 999, а рядок "999".

Якщо Ви хочете зробити математичні обчислення, використовуючи змінну b, щоб отримати цілочисельну форму змінної b, використовуйте функцію int(), щоб отримати форму змінної b з плаваючою точкою, використовуйте функцію float(), а потім збережіть це як нове значення в змінну b. Якщо Ви передасте в int() або float(), значення, яке ця функція не може оцінити як число, python відобразить повідомлення про помилку, наприклад:

```
print(int(999))
print(int('nine hundred and ninety-nine'))

Результат роботи коду:
999
Traceback (most recent call last):
File "PATH", line 37, in <module>
    print(int('nine hundred and ninety-nine'))
                        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: 'nine hundred
and ninety-nine'
```

Для друку (виведення) даних використовується функція print(). У python ця функція має параметр під назвою «end». За замовчуванням значенням цього параметра є "\n", тобто символ нового рядка, але можна завершити оператор друку будь-яким символом/рядком за допомогою цього параметра, наприклад: end=" ", end="@" тощо.

Функція print() використовує параметр «sep» для розділення аргументів, за замовчуванням значенням цього параметра є " ", тобто пробіл. Але можна використовувати будь-який символ/рядок для розділення даних за допомогою цього параметра: sep=","; sep="-"; sep="+"; sep=";" тощо.

```
print('red', 'green', 'blue', sep='-', end='\nEND\n')

Результат роботи коду:
red-green-blue
END
```

1.2.5. Операнди і оператори

У мові програмування python вирази є основним видом інструкцій. Вирази – це просто значення, об'єднані операторами, і вони завжди обчислюються до одного значення. Можна виділити дві складові виразів: значення (операнди) та дії (оператори). Наприклад, у виразі «3 + 2» – «3» та «2» є значеннями (операндами), а «+» є оператором.

Вирази можна використовувати в будь-якому місці коду python. У попередньому прикладі вираз «3 + 2» обчислюється до одного значення «5», яке також можна розглядати як вираз.

1.2.6. Типи операторів в python

У мові програмування python є такі типи операторів [13] (Рисунок 1.8):

- Арифметичні оператори
- Оператори порівняння

- Логічні оператори
- Оператори приналежності
- Оператори тотожності (ідентифікації)
- Оператори присвоєння (зв’язування)
- Побітові (порозрядні) оператори

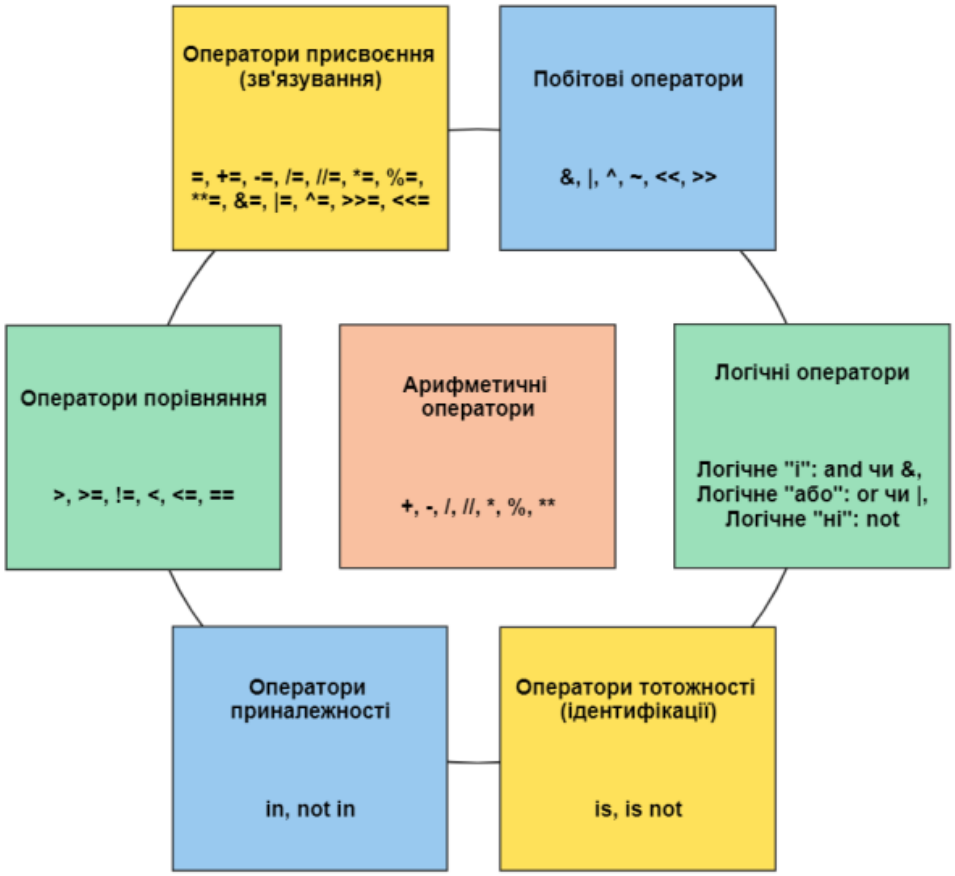


Рисунок 1.8 – Типи операторів в python

1.2.6.1. Арифметичні оператори в python

Порядок математичних операцій в python подібний до відповідного порядку в математиці. Спочатку виконується оператор ** (піднесення до степеня); оператори *, /, // (цілочисельне ділення) та % (ділення по модулю) обчислюються зліва направо; а оператори + та – оцінюються останніми також зліва направо (Таблиця 1.1). За потреби можна скористатися дужками, щоб замінити звичайний пріоритет. Пробіл між операторами та значеннями для python не має значення з точки зору інтерпретатора, але за домовленістю (стандарт оформлення коду PEP 8) потрібно використовувати один пробіл між оператором та операндом.

Таблиця 1.1 – Арифметичні оператори python

Оператор	Назва	Пояснення	Приклади
+	Додавання	Додає об’єкти	7 + 5 = 12; 'c' + 'b' дорівнює 'cb'
-	Віднімання	Повертає різницю двох чисел; якщо перший операнд відсутній, він вважається рівним нулю	-9.2 маємо від’ємне число, 46 - 21 маємо 25
*	Множення	Дає добуток двох чисел або повертає рядок, заданий числом раз, яке	6 * 3 маємо 18, 'ra' * 4 маємо 'rararara'.

		вказується після знаку множення	
**	Піднесення до степеня	Повертає число, піднесене до степеня	4 ** 5 маємо 1024 (тобто 4 * 4 * 4 * 4 * 4)
/	Ділення	Повертає частку від ділення	13 / 3 маємо 4.3333333333333333.
//	Цілочисельне ділення	Повертає неповну частку від ділення для найближчого найменшого цілого числа	7 // 3 маємо 2 10.7 // 3 маємо 3.0 -10.7 // 3 маємо -4.0
%	Ділення по модулю	Повертає залишок від ділення	16 % 5 маємо 1 -45 % 2.5 маємо 0 -45 % 2 маємо 1

Значення оператора python може змінюватися залежно від типів даних. Наприклад, + є оператором додавання, коли він працює з двома цілими числами або значеннями з рухомою комою. Однак, коли + використовується для двох рядкових значень, він використовується до рядків як оператор конкатенації рядків.

```
print(15 + 75.89)   # Операція додавання
print('Аня' + ' - студентка')   # Конкатенація рядків

Результат роботи коду:
90.89
Аня - студентка
```

Вираз і в цьому випадку обчислюється до одного значення – нового рядка, який поєднує текст двох рядків. Однак, якщо Ви спробуєте використовувати оператор + для рядка та типу цілого, python відобразить повідомлення про помилку:

```
print(100 + ' балів')

Результат роботи коду:
Traceback (most recent call last):
  File "PATH", line 48, in <module>
    print(100 + ' балів')
            ~~~~^~~~~~~
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Оператор * використовується для множення двох цілих чисел або двох чисел з рухомою комою. Але коли оператор * використовується для одного значення рядка та одного цілого значення, він стає оператором реплікації рядка, як в наступному прикладі:

```
a = 'Аня ' * 5
print(a)

Результат роботи коду:
Аня Аня Аня Аня Аня
```

Таким чином оператор * може використовуватися лише з двома числовими значеннями (для множення), або одним рядковим значенням, і одним цілим значенням (для реплікації рядків). В іншому випадку python просто відобразить повідомлення про помилку, наприклад:

```
a = 'Аня' * 'Максим'

Результат роботи коду:
Traceback (most recent call last):
  File "PATH", line 55, in <module>
```

```
    a = 'Аня' * 'Максим'
          ~~~~^~~~~~~~
TypeError: can't multiply sequence by non-int of type 'str'
```

Хоча значення рядка вважається абсолютно відмінним від цілого числа чи числа з рухомою комою, python робить це розрізнення, оскільки рядки – це текст, тоді як цілі числа та числа з рухомою комою – це числа. Але в python ціле число може бути рівним числу з рухомою комою, як в наступних прикладах:

```
12 == 12.0 – правильно (True);
12.0 == 012.000 – правильно (True);
12 == '42' – помилково (False).
```

В python = є оператором присвоєння, який надає значення правого операнда лівому. Оператор += додає значення правого операнда до лівого і надасть цю суму лівому операнду, тобто x += c рівнозначно: x = x + c, аналогічно як в Таблиці 1.2:

Таблиця 1.2 – Особливості використання арифметичних та бітових операторів в python

Оператор	Приклад	Еквівалентно
+=	x+= c	x = x + c
-=	x -= c	x = x – c
*=	x *= c	x = x * c
/=	x /= c	x = x / c
%=	x %= c	x = x % c
//=	x //= c	x = x // c
**=	x **= c	x = x ** c
&=	x &= c	x = x & c
|=	x |= c	x = x | c
^=	x ^= c	x = x ^ c
>>=	x >>= c	x = x >> c
<<=	x <<= c	x = x << c

1.2.6.2. Оператори порівняння в python

Як вже зазначалося, один знак "=" (дорівнює) застосовується для надання значення змінній. Для перевірки на рівність використовуються два знаки "дорівнює" (==) (Таблиця 1.3). Пріоритет операторів (операцій) порівняння (таблиця 3) менший за пріоритет арифметичних операцій (таблиця 1), це означає, що спочатку розраховуються результати арифметичних операцій по обидві сторони від знаку порівняння, а потім вони порівнюються. Всі оператори порівняння повертають логічний тип даних (також булів, булевий) – це простий тип даних, що може набувати двох можливих значень істина (True) або хибність

(False). Значення False не обов'язково явно означає False. Наприклад, до False прирівнюються всі перелічені значення: булева змінна False; значення None; ціле число 0; число 0.0 з плаваючою точкою; порожній рядок (' '); порожній список ([]); порожній кортеж (()); порожній словник ({}); порожня множина set (()).

Таблиця 1.3 – Оператори порівняння

Оператор		Значення	Результат
==	Рівне	Перевіряє чи обидва операнда рівні	x = 7; y = 7; x == y повертає True x = 'str'; y = 'stR'; x == y повертає False x = 'str'; y = 'str'; x == y повертає True.
<	Менше	Визначає, чи вірно, що x менше y.	7 < 3 повертає False, а 3 < 7 повертає True. Можна складати довільний ланцюжок порівнянь: 3 <7 <11 дає True.
>	Більше	Визначає, чи вірно, що x більше y	x = 7; y = 5; 7 > 3 повертає True.
<=	Менше або рівне	Визначає, чи вірно, що x менше або дорівнює y	x = 7; y = 9; x <= y повертає True
>=	Більше або рівне	Визначає, чи вірно, що x більше або дорівнює y	x = 8; y = 3; x >= 3 повертає True
!=	Не рівне	Перевіряє, чи вірно, що об'єкти не рівні	x = 8; y = 5; x != y повертає є True

Рядки в python теж можна порівнювати по аналогії з числами. Символи, як і все інше, представлено в комп'ютері у вигляді чисел. Є спеціальна таблиця, яка ставить у відповідність кожному символу деяке число. Визначити, яке число відповідає символу, можна за допомогою функції ord(), наприклад:

```
print(f"ord('r') = {ord('r')}, ord('d') = {ord('d')}, ord('a') =
{ord('a')}")
print(f"'r' == 'd' - {'r' == 'd'}, 'r' > 'd' - {'r' > 'd'}, 'r' +
'd' > 'r' - {'r' + 'd' > 'r'}")
результат роботи коду:
ord('r') = 114, ord('d') = 100, ord('a') = 97
'r' == 'd' - False, 'r' > 'd' - True, 'r' + 'd' > 'r' - True
```

1.2.6.3. Логічні оператори в python (and, or, not)

Python надає логічні оператори (Таблиця 1.4) для виконання логічних операцій (and, or, not), для опису зв'язку між кількома умовами в операторі if, для об'єднання простих виразів в більш складні. Значення, що повертаються при використання логічних операторів мають тип bool - True або False. Значення, що повертаються при використання логічних операторів and, or, не обов'язково мають тип bool, як в наступному прикладі:

```
a2 = input('Американський біолог – один із основоположників
генетики\n')
if (a2 == 'Томас Хант Морган') or (a2 == 'Томас Морган') or (a2 ==
'Морган') :
   print('Відповідь вірна\n')
else:
   print('Відповідь не правильна\n')

x = True
y = True
```

```
print(f'x = True y = True, x and y = {x and y}')
x1 = True
y1 = False
print('x1 = True y1 = False, x1 and y1 = ', x1 and y1)
x2 = 0
y2 = 100
print(f'x2 = 0 y2 = 100, x2 and y2 =', x2 and y2)
x3 = 0.3
y3 = 100
print(f'x3 = 0.3 y3 = 100, x3 and y3 = ', x3 and y3)
x4 = 1
y4 = 100
print(f'x4 = 1 y4 = 100, 4 and y4 = ', x4 and y4)
x5 = 1
y5 = 0
print(f'x5 = 1 y5 = 0, x5 and y5 = ', x5 and y5)
x6 = -1
y6 = 25
print(f'x6 = -1 y6 = 25, x6 and y6 = ', x6 and y6)
x = True
y = True
print('x = True y = True, x or y  =  ', x or y)
x = True
y = False
print('x = True y = False, x or y1 =  ', x or y1)
x = False
y = False
print('x = False y = False, x or y  =  ', x or y)
x = True
print('x = True, not x = ', not x)
y = False
print('y = False, not y = ', not y)
```

Результат роботи коду:

```
Американський біолог – один із основоположників генетики
Морган
Відповідь вірна
x = True y = True, x and y = True
x1 = True y1 = False, x1 and y1 =  False
x2 = 0 y2 = 100, x2 and y2 = 0
x3 = 0.3 y3 = 100, x3 and y3 =  100
x4 = 1 y4 = 100, 4 and y4 =  100
x5 = 1 y5 = 0, x5 and y5 =  0
x6 = -1 y6 = 25, x6 and y6 =  25
x = True y = True, x or y  =   True
x = True y = False, x or y1 =   True
x = False y = False, x or y  =   False
x = True, not x =  False
y = False, not y =  True
```

Таблиця 1.4 – Логічні оператори по пріоритетності їх виконання

Оператор		Значення	Результат
and	Логічне і (перехрестя)	x = True, y = True x = False; y = True x = 0; y = 10 x = 1; y = 5 x = 15; y = 5	x and y повертає True x and y повертає False, в цьому випадку python не перевіряє значення y, так як вже ліва частина виразу 'and' дорівнює False, це називається скороченою оцінкою булевих (логічних) виразів. x and y повертає 0, тобто False x and y повертає 5, тобто True x and y повертає 15, тобто всі значення, що не дорівнюють 0, сприймаються як True і python повертає більше з двох чисел.
or	Логічне або (доповнення)	x = True y = True, x = True y = False x = False y = False	x or y повертає True x or y повертає True x or y повертає False
not	Логічне ні (заперечення)	x = True y = False,	not x повертає False not y повертає True

1.2.6.4. Оператор приналежності in в python

Оператор приналежності in в python перевіряє входження підрядка в рядок, наприклад:

```
q = 'ab' in 'abccda'
print("Приналежність підрядка 'ab' рядку 'abccda' - ", q)
q = 'A' in 'abccda'
print("Приналежність символу 'A' рядку 'abccda' - " , q)
print(f"'12' in '12345' - {'12' in '12345'}")
print(f"'15' in '12345' - {'15' in '12345'}")
Результат роботи коду:
Приналежність підрядка 'ab' рядку 'abccda' -  True
Приналежність символу 'A' рядку 'abccda' -  False
'12' in '12345' - True
'15' in '12345' - False
```

1.2.6.5. Оператори тотожності (ідентифікації) is та is not в python

Оператори тотожності (ідентифікації) is та is not в python використовуються для перевірки, чи дві змінні вказують на єдиний об’єкт у пам’яті. Якщо дві змінні дорівнюють одна одній (оператор ==), то це не означає, що вони ідентичні, наприклад:

```
colors = ['red', 'green', 'blue']
colors1 = colors
colors2 = colors.copy()
print(colors1 == colors)
print(colors2 == colors)
print(colors1 is colors)
```

```
print(colors2 is colors)

Результат роботи коду:
True
True
True
False
```

У першому випадку new_colors є лише синонімом colors, який вказує на ідентичний об’єкт. Проте при створенні копії було створено інший об’єкт у пам’яті, на що вказує оператор is.

1.2.6.6. Бітові (порозрядні) оператори в python

Бітові або порозрядні оператори в мові python підтримують роботу з двійковими розрядами (бітами) цілочисельних величин, де кожен біт числа розглядається окремо. Ці оператори реалізують загальновідомі бітові операції. Підтримка бітових операторів є також в інших мовах програмування.

У бітових операторах (операціях) кожен операнд (значення) розглядається як послідовність двійкових розрядів (бітів), що приймають значення 0 або 1 (двійкова система числення).

Перелік бітових операторів мови python в порядку спадання пріоритету такий:

- ~ – бітовий оператор НІ (інверсія, найвищий пріоритет);
- <<, >> – оператори зсуву вліво або зсуву вправо на задану кількість біт;
- & – бітовий оператор І (AND);
- ^ – бітове виключне АБО (XOR);
- | – бітовий оператор АБО (OR).

Бітовий оператор ~ (НІ). Результат застосування побітової інверсії розраховується за формулою: ~x = -(x + 1). Наприклад, print(~1011) в результаті дає негативне число -1012, а print(~-1011) в результаті дає позитивне число 1010.

Оператори зсуву вліво << та зсуву вправо >> зсувають кожен біт на одну або декілька позицій вліво чи вправо. Послідовність застосування операторів зсуву наступна. Спочатку число, яке стоїть ліворуч від оператора зсуву, наприклад, 5 переводиться у двійкову систему. Потім кожен біт зсувається ліворуч для оператора зсуву вліво << та праворуч для оператора зсуву вправо >> на ту кількість позицій, яка дорівнює числу, яке стоїть праворуч від оператора зсуву, наприклад, на 1 позицію для виразу 5<<1 (Рисунок 1.9). Потім отриманий результат переводиться із двійкової системи в десятинну, саме це число і є результатом застосування оператору зсуву. Для наведеного прикладу, в результаті отримаємо число 10.

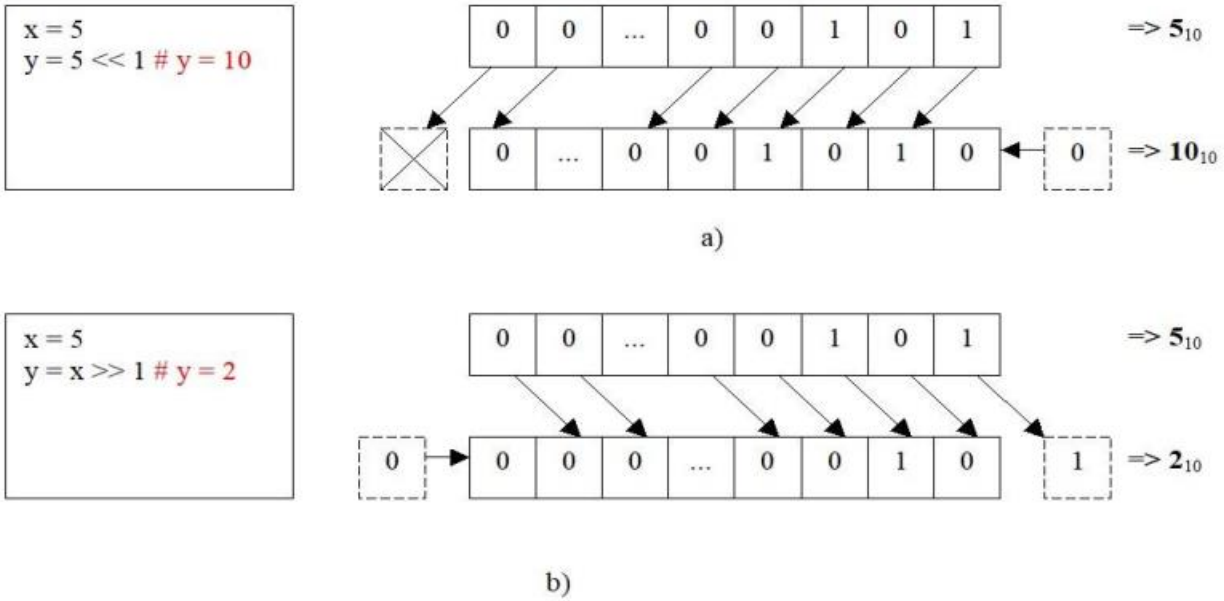


Рисунок 1.9 – Схема застосування бітових операторів зсуву вліво << та зсуву вправо >> .

Бітовий оператор & – І (AND). Кожен цілочисельний операнд розглядається як набір бітів, над кожним з яких виконується побітова операція “&” (І). Тобто спочатку обидва числа переводяться у двійкову систему, а потім результат обчислюється за такою логічною таблицею:

0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1

Приклад застосування бітового оператора “&” наведено на Рисунку 1.10, а саме для 188&80. В результаті отримуємо число 16 (після переведення із двійкової в десятинну систему).

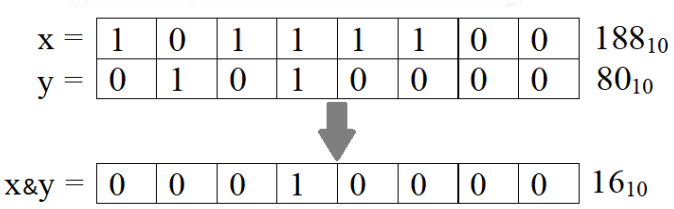


Рисунок 1.10 – Схема реалізації бітового оператору & “І” для 188&80.

Оператор “^” – виключне АБО (XOR) також оперує з двійковими розрядами. Кожен операнд розглядається як послідовність бітів. Результат побітового виключного АБО визначається за такими формулами:

0 ^ 0 = 0
0 ^ 1 = 1
1 ^ 0 = 1
1 ^ 1 = 0

На Рисунку 1.11 відображено приклад бітового виключного АБО для двох операндів для прикладу 188^80. В результаті отримуємо число 236.

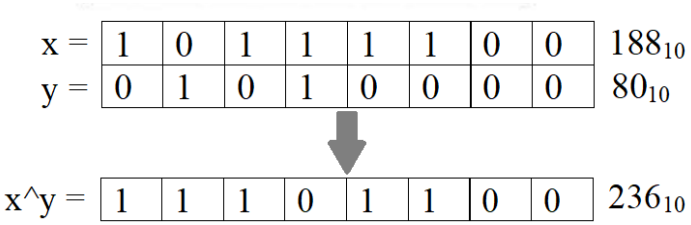


Рисунок 1.11 – Схема застосування бітового оператору ^ “виключне АБО” для прикладу 188^80=236.

Бітовий оператор АБО (OR) є бінарним і позначається символом |. Оператор реалізує побітове логічне додавання.

Бітовий оператор АБО виконується за такими правилами:

0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1

На Рисунку 1.12 продемонстровано роботу бітового оператора | (АБО) для прикладу 188|80. В результаті отримуємо число 252.

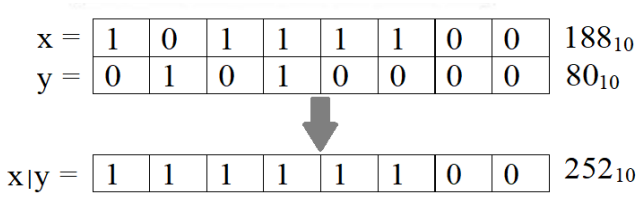


Рисунок 1.12 – Схема реалізації бітового оператору АБО | для прикладу 188|80

Підсумовуючи, зазначимо, що в мові python бітові (порозрядні) оператори, призначені для роботи з даними у бітовому (двійковому) форматі. Тому спочатку числа переводяться у двійкову систему. Після чого до чисел, записаних в двійковій системі, застосовується конкретний побітовий оператор. В кінці результат переводиться із двійкової в десятинну систему.

1.2.7. Керуючі структури (оператори управління потоком)

В усіх мовах програмування високого рівня є можливість розгалуження програми; при цьому виконується одна з гілок програми в залежності від істинності чи хибності умови. В python є три керуючі структури або оператори управління потоком: умовний оператор if, оператор for і оператор while. які вимагають певного механізму для групування операторів у складені оператори або блоки та змінюють потік керування програмою – порядок виконання операторів програми. Ці структури полегшують ітерацію , багаторазове виконання оператора або блоку операторів.

Структури if, for і while подібні, як і, наприклад, в мовах C, Pascal тощо, але python використовує двокрапки і відступи для позначення блоків коду всередині оголошених керуючих структур. Двокрапка показує, що далі йде блок виразів.

1.2.7.1. Умовний оператор if

Оператор if має конструкції if-else, if-elif-else (Таблиця 1.5). Вираз elif, який поєднує в собі два пов’язаних if else - if else вирази в один вираз if-elif-else. Це полегшує читання програми, а також не вимагає додаткових відступів. Вирази elif і else також мають двокрапку в кінці логічного рядка. В кінці умовної конструкції if додаткова умова else є

необов'язковою. Якщо жодна умова не приймає значення True, буде виконаний блок після заключного else, якщо додаткова умова else в кінці умовної конструкції if відсутня, виконається наступна команда коду.

Таблиця 1.5 – Умовні оператори if

Вираз	Опис
if <умова>: <блок>	Найпростіша умовна конструкція. Якщо <умова> істинна, то <блок> виконується, в іншому випадку – пропускається.

if <умова >:
  <блок 1>
else:
  <блок 2>
	Умовна конструкція. Якщо <умова> істинна, то виконується <блок 1>, якщо хибна - <блок 2>.

if <умова 1>:
  <блок 1>
elif <умова 2>
  …
elif <умова N>:
  <блок N>
else <блок N+1>
	Умовна конструкція з додатковою умовою elif і необов'язковим else в кінці. Буде виконаний блок після першої ж умови, яка виявиться істинною. Якщо жодна умова не приймає значення True, буде виконаний блок після заключного else.

Приклад коду з конструкціями умовного оператору if:

```
score = int(input("Введіть Вашу оцінку: "))
if score >= 95:
    print("Відмінно! Ваша оцінка A")
elif score >= 85:
    print("Дуже добре! Ваша оцінка - B")
elif score >= 75:
    print("Добре! Ваша оцінка - C")
elif score >= 65:
    print("Задовільно. Ваша оцінка - D")
elif score >= 60:
    print("Достатньо. Ваша оцінка - E")
else:
    print("Ви не здали екзамен.")
```

```
      Результат виконання коду (перший варіант):
Введіть Вашу оцінку: 80
Добре! Ваша оцінка - C
      Результат виконання коду (другий варіант):
Введіть Вашу оцінку: 45
Ви не здали екзамен.
```

Приклад коду з конструкціями умовного оператору if:

```
radius = float(input("Введіть радіус кола: "))
if radius >= 0:
    print("Радіус кола = ",  2  *  3.14  *  radius)
    print("Площа кола = ", 3.14 * radius ** 2)
else:
    print("Введіть позитивне число")
```

```
      Результат виконання коду (перший варіант):
Введіть радіус кола: 3.2
Довжина кола =  20.096000000000004
Площа кола =  32.153600000000004
Результат виконання коду (другий варіант):
Введіть радіус кола: -1
Введіть позитивне число
```

1.2.7.2. Оператор циклу (інструкція) while

В мові програмування python є оператори циклів for та while. Цикли розділяються на цикли з попередньою перевіркою умови (for, while) та на цикли із наступною перевіркою умови (наприклад, while True, або while else). В циклах з попередньою перевіркою умови умова виходу із циклу перевіряється до виконання команд, заданих в «тілі циклу». В другому випадку умова виходу із циклу перевіряється після виконання команд, заданих в «тілі циклу», тому в для цикл із наступною перевіркою умови виходу із циклу один раз точно виконується. Розглянемо спочатку цикл while.

Оператор (інструкція) while дозволяє багаторазово виконувати блок команд до тих пір, поки виконується деяка умова. Це один з так званих операторів циклу. Він також може мати необов'язковий вираз else. Блок else виконується тоді, коли умова циклу while стає хибною (False) – це може статися навіть при найпершій перевірці умови. Якщо у циклу while є додатковий блок else, він завжди виконується, якщо тільки цикл не буде перерваний оператором break або оператором continue.

Приклад коду з оператором while з попередньою перевіркою умови виходу із циклу:

```
a = 0
while a < 5:
    print('Цикл виконано', a, 'раз(ів)')
    a = a + 1
else:
    print('Цикл закінчено')
```

```
Результат виконання коду:
Цикл виконано 1 раз(ів)
Цикл виконано 2 раз(ів)
Цикл виконано 3 раз(ів)
Цикл виконано 4 раз(ів)

Цикл закінчено
```

Нижче наведено приклад коду з оператором while із наступною перевіркою умови виходу із циклу. Цей цикл приймає дані користувача за допомогою вбудованої функції input(). Потім вхідні дані перетворюються на ціле число за допомогою int(). Якщо користувач вводить число, що дорівнює 0 або менше, виконується оператор break і цикл завершується.

```
while True:
    number = int(input("Введіть позитивне число: "))
    print(number)
    if not number > 0:
        break
```

Результат роботи коду

```
Введіть позитивне число: 1
1
Введіть позитивне число: 4
4
Введіть позитивне число: -1
-1
```

1.2.7.3. Оператор циклу (інструкція) for

Оператор for також є одним з операторів циклу. Блок else після оператору for не є обов'язковим. Якщо він присутній, то завжди виконується один раз після закінчення циклу for, якщо тільки не вказано оператор break або оператор continue.

Цикл for має такий синтаксис:

for <variable> in range(<number>):

for <variable> in range(<start_number>, <end_number>):

for <variable> in range(<start_number>, <end_number>, <step_size>):

Оператор for..in також є оператором циклу, який здійснює ітерацію по послідовності об'єктів, тобто проходить через кожен елемент в послідовності. Послідовність – це упорядкований набір елементів. Варто звернути увагу, що функція range () генерує послідовність чисел, але тільки по одному числу за раз – коли оператор for запитує наступний елемент.

Щоб побачити всю послідовність чисел відразу, використовується функція list(range ()). Потім цикл for здійснює ітерацію з заданого діапазону: for i in range (1, 5) еквівалентно for i in [1, 2, 3, 4], що нагадує присвоювання змінній і по одному числу (або об'єкту) за раз, виконуючи блок команд для кожного значення і. В даному прикладі в блоці команд просто виводимо значення в консоль. Цикл for..in працює для будь-якої послідовності. У даному прикладі це список чисел, згенерований вбудованою функцією python range(), але в загальному випадку можна використовувати будь-яку послідовність будь-яких об'єктів.

Як вже зазначалося, python надає два ключові слова (оператори), які передчасно завершують ітерацію циклу: оператор break та оператор continue.

Оператор (ключове слово) break служить для переривання циклу, тобто зупинення виконання команд навіть якщо умова виконання циклу ще не прийняла значення False або послідовність елементів не закінчилася. Виконання програми переходить до першого оператора, наступного за тілом циклу.

Приклад коду з оператором break:

```
for i in range(0, 4):
   if i == 3:
      break
   print(i)

Результат виконання коду:
0
1
2
```

Оператор (ключове слово) continue використовується для вказівки python, що необхідно пропустити всі команди, які наразі в поточному блоці циклу for або while і продовжити виконання коду з наступної ітерації циклу. Виконання переходить до початку циклу, а керуючий вираз for або while повторно визначає, чи буде цикл виконано знову чи завершиться, тобто continue перериває поточну ітерацію і передає управління на початок циклу, після чого умова знову перевіряється, якщо умова істинна, виконується наступна ітерація.

Приклади коду з оператором continue:

```
a = 0
while a < 5:
    a += 1
    if a == 3:
      continue
    print(a)
Результат виконання коду:
1
2
4
```

Завдання та запитання до підрозділу «Базові поняття мови python: базовий синтаксис, типи даних, змінні, введення та виведення даних, математичні операції, логічні оператори, умовні оператори, цикли»

1. Дайте визначення, що таке відкрите у доступі програмне забезпечення.
2. Які символи може містити ідентифікатор (назва) змінних?
3. Відмітьте ідентифікатори (назви) змінних, які є коректними: f1, a_b_95, _ab_a, 2a, 10_, A1, Sxy.
4. Знайдіть помилки в коді:

```
a = input('Введіть число '’)
b = input('Введіть число ')
d = a**b – 10.0
c = -
c = c*20
print(d,'\n', c)
```

5. Відмітьте оператори порівняння серед наведених нижче операторів: == , != , >, < , <= , >= , in, not in, is, is not.
6. Відмітьте оператори циклу серед наведених нижче операторів: for…in, while, if else, if elif, for…in.
7. Напишіть код, який обчислює суму всіх чисел від 1 до заданого числа і виводить її в консоль, якщо сума менше 100 мільйонів. Якщо сума більша за 100 мільйонів, вивести в консоль повідомлення «Сума перевищує 100 мільйонів».
8. Напишіть код на мові Python для пошуку чисел, які діляться на 7 без залишку та кратні 5, між 1500 і 2 000 000 обидва включені). Розрахуйте і виведіть в консоль всі знайдені числа, загальну кількість знайдених чисел та знайдене число з заданим номером по порядку.

1.3. Складні типи даних python – рядки, списки, кортежі, словники, множини

1.3.1. Тип даних рядки (str)

Рядки в python – це впорядковані послідовності символів, які представляють текстові дані. Рядки є незмінними та можуть містити будь-які символи, що підтримуються стандартом Unicode.

Рядки можна створювати багатьма способами:

- Використання літерала – парної кількості лапок:
  - Одинарні лапки: ''
  - Подвійні лапки: ""
  - Потрійні лапки (одинарні або подвійні): """ або """"""""
- Використання літерала – парної кількості лапок:
  - Одинарні лапки: 'abc'. У рядку, який створено за допомогою одинарних лапок, можна використовувати символ подвійних лапок без виникнення помилки.

- Подвійні лапки: "abc". У рядку, який створено за допомогою подвійних лапок, можна використовувати символ одинарних лапок без виникнення помилки.
- Потрійні лапки (одинарні або подвійні): '''abc''' або """abc""". Рядки з потрійними лапками можуть охоплювати кілька рядків при цьому всі переходи на новий рядок будуть збережені у вигляді спеціальних екранованих послідовностей ('\n').

• Використання вбудованої в python функції: str(), якщо аргумент у функції str() не задано, створюється новий порожній рядок. За допомогою цієї функції можна перетворити дані будь-якого типу на рядок.

Приклади коду створення рядків:

```
print("Створення рядків:")
# створюємо рядок за допомогою:
# літерала
# одинарні лапки
str0 = ''
print('str0 = ', str0)
str1 = 'abcd'
print('str1 = ', str1)
# подвійні лапки
str2 = ""
print('str2 = ', str2)
str3 = "abcd"
print('str3 = ', str3)
# потрійні лапки
str4 = ''''''
print('str4 = ', str4)
str5 = """a,
b,
c,
d."""
print('str5 = ', str5)
# вбудованої функції str()
str6 = str({456447: 'abc'})
print('str6 = ', str6)

Результат роботи коду:
Створення рядків:
str0 =
str1 = abcd
str2 =
str3 = abcd
str4 =
str5 = a,
b,
c,
d.
str6 = {456447: 'abc'}
```

Отримати доступ до символів в рядку можна за допомогою оператора нарізки ([] і [:]) і індексів, починаючи з нуля і до кінця списку:

```
str1 = 'Python'
```

```
# Друк всього списку str1
print(f'{str1 = }')
# Друк першого символу рядка str1
print(f'{str1[0] = }')
# Друк останнього символу рядка str1
print(f'{str1[-1] = }')
# Друк перших двох символів рядку str1
print(f'{str1[:2] = }')
# Друк всіх символів у рядку str1
print(f'{str1[:] = }')
# Друк від 2-го символу до 4-го, не включаючи останній, з кроком 2
print(f'{str1[2:5:2] = }')
```

```
Результат роботи коду:
str1 = 'Python'
str1[0] = 'P'
str1[-1] = 'n'
str1[:2] = 'Py'
str1[:] = 'Python'
str1[2:5:2] = 'to'
```

Форматовані рядки у python є важливим інструментом для вставки значень змінних в текстові рядки з використанням спеціального синтаксису, такого як фігурні дужки {}. Вони дозволяють створювати зрозумілі та читабельні рядки, що містять дані з різних змінних без необхідності складання рядка вручну:

```
language = 'Python'
version = 3.11
print(f'Мова програмування {language}. Версія {version}')
print(f'{language = }')   # Простий спосіб виведення назви змінної та значення
```

```
Результат роботи коду:
Мова програмування Python. Версія 3.11
language = 'Python'
```

Якщо перед рядком вказано літеру r, то такі рядки називаються "сирі" (raw strings). Вони використовуються для того, щоб уникнути екранування спеціальних символів, таких як \n або \t. Сирі рядки вказують інтерпретатору ігнорувати спеціальні символи у тексті рядка і трактувати його як звичайний текст. Це особливо корисно, наприклад, при роботі з регулярними виразами, шляхами файлів у Windows (оскільки там використовується \ як роздільник шляхів), або при обробці даних, де спеціальні символи повинні бути збережені в незмінному вигляді:

```
print(r'C:\Users\teacher\Desktop\newProject\venv\Scripts\python.exe')
```

```
Результат роботи коду:
C:\Users\admin\Desktop\pythonProject\venv\Scripts\python.exe
```

Розглянемо методи роботи із рядками:

str1.isalpha() – повертає True, якщо непорожній рядок складається лише з алфавітних символів.

str1.isnumeric() – повертає True, якщо непорожній рядок містить будь-які цифри (у тому числі розпізнає римські цифри).
str1.isalnum() – повертає True, якщо непорожній рядок містить будь-які алфавітні символи або цифри.
str1.isdigit() – повертає True, якщо всі символи непорожнього рядка – цифри, дроби чи верхні індекси.
str1.isdecimal() – повертає True, якщо непорожній рядок містить символи, які формують число у десятковій системі числення.
str1.isascii() – повертає True, якщо непорожній рядок містить лише символи, які є в системі кодів ASCII.
str1.lower() – змінює регістр всіх символів на нижній.
str1.casefold() – аналогічно до str1.lower() змінює регістр всіх символів на нижній, але більш агресивно. Наприклад, німецька мала літера "ß" еквівалентна "ss". Оскільки це вже нижній регістр, lower() нічого не зробить для "ß"; casefold() перетворює його на "ss".
str1.upper() – змінює регістр всіх символів на верхній.
str1.title() – змінює регістр початкових символів всіх слів у рядку на верхній, а всі інші символи – на нижній регістр.
str1.capitalize() – змінює у верхній регістр першу літеру тільки першого слова рядка, а всі інші перетворює на нижній регістр.
str1.islower() – повертає True, якщо непорожній рядок складається лише із символів у нижньому регістрі.
str1.isupper() – повертає True, якщо всі символи непорожнього рядка у верхньому регістрі.
str1.istitle() – повертає True, якщо рядок оформлено як заголовок і містить принаймні один символ. Метод перевіряє, щоб символи верхнього регістру слідувати лише за символами без регістру, а символи нижнього регістру – лише за символами з регістром. В іншому випадку повертає False.
str1.isprintable() – повертає True, якщо всі символи в рядку придатні для друку або рядок порожній, False в іншому випадку. Недруковані символи – це символи, визначені в базі даних символів Unicode як «Інші» або «Роздільники», за винятком пробілу.
str1.isspace() – повертає True, якщо в рядку є лише пробіли та є принаймні один символ, інакше False.
str1.isidentifier() – повертає True, якщо рядок можна використати як назву змінної відповідно до правил їх найменування, інакше False.
str1.startswith(str) – повертає True, якщо рядок починається з підрядка str1.
str1.endswith(str) – повертає True, якщо рядок закінчується на підрядок str1.
str1.removeprefix(prefix) – Якщо рядок починається з вказаного префікса, повертається зріз str[len(prefix):]. В іншому випадку повертається копія вихідного рядка.
str1.removesuffix(suffix) – якщо рядок закінчується на вказаний суфікс, і цей суфікс не порожній, повертається зріз str1[:-len(suffix)]. В іншому випадку повертається копія вихідного рядка.
str1.lstrip() – видаляє початкові пробіли з рядка.
str1.rstrip() – видаляє кінцеві пробіли з рядка.
str1.strip() – видаляє початкові та кінцеві пробіли з рядка.
str1.expandtabs(tabsize=8) – повертає копію рядка, де всі символи табуляції замінено одним або кількома пробілами, залежно від поточного стовпця та заданого розміру табуляції.
str1.ljust(width) – якщо довжина рядка менша за параметр width, то праворуч від рядка додаються пробіли, щоб довжина рядка складала width. Якщо довжина рядка більша за width, то рядок повертається без змін.
str1.rjust(width) – якщо довжина рядка менша за параметр width, то ліворуч від рядка додаються пробіли, щоб довжина рядка складала width. Якщо довжина рядка більша за width, то рядок повертається без змін.

str1.center(width) – якщо довжина рядка менша за параметр width, то ліворуч і праворуч від рядка рівномірно додаються пробіли, щоб доповнити значення width, а сам рядок вирівнюється по центу. Якщо довжина рядка більша за width, то рядок повертається без змін.
str1.zfill(width) – повертає копію рядка зліва, заповненого цифрами '0', щоб створити рядок довжини width. Якщо перший символ рядка '+' чи '-', то нулі заповнюються даних символів знака, а не перед ним. Якщо значення width менше або дорівнює довжині рядка, то повертається рядок без змін.
str1.find(str[, start[, end]) – повертає найменший індекс входження підрядка у рядку str1. Якщо підрядок не знайдено, повертається число -1.
str.rfind(str[, start[, end]) – повертає найбільший індекс входження підрядка у рядку str1. Якщо підрядок не знайдено, повертається число -1.
str1.index(str[, start[, end]) – повертає найменший індекс входження підрядка у рядку str1. Якщо підрядок не знайдено, то виникає помилка ValueError.
str.rindex(str[, start[, end]) – повертає найбільший індекс входження підрядка у рядку str1. Якщо підрядок не знайдено, то виникає помилка ValueError.
str1.split(sep=None, maxsplit=- 1) – розбиває рядок на підрядки в залежності від роздільника sep, повертає список рядків. Якщо задано maxsplit, виконується не більше ніж maxsplit, починаючи розділення зліва.
str.rsplit(sep=None, maxsplit=-1) – розбиває рядок на підрядки в залежності від роздільника sep, повертає список рядків. Якщо задано maxsplit, виконується не більше ніж maxsplit, починаючи розділення справа.
str1.splilines(keepend=False) – повертає список рядків у рядку, розриваючи межі рядків. Розриви рядків не включаються в результуючий список, якщо значення keepend задано False. Межами рядків вважаються символи: \n, \r, \r\n, \v (\x0b), \f (\x0c), \x1c, \x1d, \x1e, \x85, \u2028, \u2029.
str1.partition(sep) – розділяє рядок за першим входженням підрядка sep і повертає кортеж із 3 елементів: частина перед роздільником, сам роздільник і частина після роздільника. Якщо роздільник не знайдено, повертається кортеж із трьох елементів, що містить сам рядок, а потім два порожні рядки.
str1.rpartition(sep) – розділяє рядок за останнім входженням підрядка sep і повертає кортеж із 3 елементів: частина перед роздільником, сам роздільник і частина після роздільника. Якщо роздільник не знайдено, повертається кортеж із трьох елементів, що містить сам рядок, а потім два порожні рядки.
str1.replace(old, new[, num]) – замінює в рядку один підрядок на інший.
str1.join(iter) – об’єднує рядки зі списку iter (або іншого об’єкта, який підтримує ітерацію) в один рядок, вставляючи між ними вказаний роздільник str1.
str.count(sub[, start[, end]]) – повертає кількість входжень підрядка sub без перекриття у діапазоні [start, end]. Необов'язкові аргументи початок і кінець інтерпретуються як у нотації фрагментів.
str.encode(encoding='utf-8', errors='strict') – повертає рядок, закодований у байтах. Кодування за замовчуванням 'utf-8'; дивіться стандартні кодування для можливих значень. Параметр errors контролює, як обробляються помилки кодування. Якщо "strict" (за замовчуванням), виникає виняток UnicodeError. Інші можливі значення: «ignore», «replace», «xmlcharrefreplace», «backslashreplace» та будь-яке інше ім’я, зареєстроване через codecs.register_error().
str.format(*args, **kwargs) – виконує операцію форматування рядка. Рядок, у якому викликається цей метод, може містити літеральний текст або поля заміни, розділені дужками {}. Кожне поле заміни містить або числовий індекс позиційного аргументу, або назву ключового аргументу. Повертає копію рядка, де кожне поле заміни замінено рядковим значенням відповідного аргументу. З появою форматованих рядків (f-strings) метод втратив актуальність.

str.format_map(mapping) – подібно до str.format(**mapping), за винятком того, що відображення використовується безпосередньо, а не копіюється в dict.
str.translate(table) – повертає копію рядка, у якому кожен символ було зіставлено через дану таблицю table. Таблиця має бути об’єктом, який реалізує індексування через __getitem__(), як правило, відображення або послідовність. Можна використовувати str.maketrans(), щоб створити карту перекладу з відображень символів у різні формати.
str.maketrans(x[, y[, z]]) – повертає таблицю перекладу, яку можна використовувати для str.translate(). Якщо є лише один аргумент, це має бути словник, який відображає порядкові номери Unicode (цілі числа) або символи (рядки довжиною 1) на порядкові номери Unicode, рядки (довільної довжини) або None. Потім символьні ключі будуть перетворені на порядкові. Якщо є два аргументи, вони мають бути рядками однакової довжини, і в отриманому словнику кожен символ у x буде зіставлено зі символом у тій же позиції в y. Якщо є третій аргумент, це має бути рядок, символи якого будуть відображені в результаті як «None».

Приклади методів роботи із рядками – str.isalpha(), str.isnumeric(), str.isalnum(), str.isdigit(), str.isdecimal(), str.isascii():

```
# створюємо об'єкти - рядки str1, str2, str3 та str4
str1 = 'Python'
print(f'{str1 = }')
str2 = '3'
print(f'{str2 = }')
str3 = 'Ⅲ'
print(f'{str3 = }')
str4 = str1 + str2
print(f'{str4 = }')
print(str1.isalpha(), str2.isalpha(), str3.isalpha(),
str4.isalpha())
print(str1.isnumeric(), str2.isnumeric(), str3.isnumeric(),
str4.isnumeric())
print(str1.isalnum(), str2.isalnum(), str3.isalnum(),
str4.isalnum())
print(str1.isdigit(), str2.isdigit(), str3.isdigit(),
str4.isdigit())
print(str1.isdecimal(), str2.isdecimal(), str3.isdecimal(),
str4.isdigit())
print(str1.isascii(), str2.isascii(), str3.isascii(),
str4.isascii())

Результат роботи коду:
str1 = 'Python'
str2 = '3'
str3 = 'Ⅲ'
str4 = 'Python3'
True False False False
False True True False
True True True True
False True False False
False True False False
True True False True
```

Приклади роботи методів із рядками – str.lower(), str.casefold(), str.upper(), str.title(), str.capitalize():

```
str1 = 'python languAGE'
print(f'{str1 = }')
str2 = 'Fußball'
print(f'{str2 = }')
print(f'{str1.lower() = }')
print(f'{str2.lower() = }')
print(f'{str1.casefold() = }')
print(f'{str2.casefold() = }')
print(f'{str1.upper() = }')
print(f'{str1.title() = }')
print(f'{str1.capitalize() = }')

Результати роботи коду:
str1 = 'python languAGE'
str2 = 'Fußball'
str1.lower() = 'python language'
str2.lower() = 'fußball'
str1.casefold() = 'python language'
str2.casefold() = 'fussball'
str1.upper() = 'PYTHON LANGUAGE'
str1.title() = 'Python Language'
str1.capitalize() = 'Python language'
```

Приклади методів роботи із рядками – str.islower(), str.isupper(), str.istitle():

```
str1 = 'New Python'
print(f'{str1 = }')
str2 = 'KPI'
print(f'{str2 = }')
str3 = 'string'
print(f'{str3 = }')
print(str1.islower(), str2.islower(), str3.islower())
print(str1.isupper(), str2.isupper(), str3.isupper())
print(str1.istitle(), str2.istitle(), str3.istitle())

Результат роботи коду:
str1 = 'New Python'
str2 = 'KPI'
str3 = 'string'
False False True
False True False
True False False
```

Приклади роботи методів із рядками – str.isprintable(), str.isspace(), str.isidentifier():

```
str1 = '     '
print(f'{str1 = }')
str2 = '_print'
print(f'{str2 = }')
```

```
str3 = '3print'  # назва змінної не може починатись із цифри
print(f'{str3 = }')
str4 = '\n'
print(f'{str4 = }')
print(str1.isprintable(),  str2.isprintable(),  str3.isprintable(),
str4.isprintable())
print(str1.isspace(),        str2.isspace(),        str3.isspace(),
str4.isspace())
print(str1.isidentifier(),                    str2.isidentifier(),
str3.isidentifier(), str4.isidentifier())

Результат роботи коду:
str1 = '     '
str2 = '_print'
str3 = '3print'
str4 = '\n'
True True True False
True False False True
False True False False
```

Приклади роботи методів із рядками – str.startwith(), str.endwith(), str.removeprefix(), str.removesuffix():

```
str1 = 'uncomfortable'
print(f'{str1 = }')
str2 = 'comfort'
print(f'{str2 = }')
print(str1.startswith('un'), str2.startswith('un'))
print(str1.endswith('able'), str2.endswith('able'))
print(str1.removeprefix('un'), str2.removeprefix('un'))
print(str1.removesuffix('able'), str2.removesuffix('able'))

Результат роботи коду:
str1 = 'uncomfortable'
str2 = 'comfort'
True False
True False
comfortable comfort
uncomfort comfort
```

Приклади роботи методів із рядками – str.lstrip(), str.rstrip(), str.strip(), str.expandtabs():

```
str1 = '   Python   '
print(f'{str1 = }')
str2 = '\tPython\tCode\t'
print(f'{str2 = }')
print(f'{str1.lstrip() = }')
print(f'{str1.rstrip() = }')
print(f'{str1.strip() = }')
print(f'{str2.strip() = }')
print(f'{str2.expandtabs(tabsize=5) = }')
```

```
Результат роботи коду:
str1 = '   Python   '
str2 = '\tPython\tCode\t'
str1.lstrip() = 'Python   '
str1.rstrip() = '   Python'
str1.strip() = 'Python'
str2.strip() = 'Python\tCode'
str2.expandtabs(tabsize=5) = '     Python    Code '
```

Приклади роботи методів із рядками – str.ljust(), str.rjust(), str.center(), str.zfill():

```
str1 = 'Python'
print(f'{str1 = }')
print(f'{str1.ljust(10) = }')
print(f'{str1.rjust(10) = }')
print(f'{str1.center(10) = }')
print(f'{str1.zfill(10) = }')

Результат роботи коду:
str1 = 'Python'
str1.ljust(10) = 'Python    '
str1.rjust(10) = '    Python'
str1.center(10) = '  Python  '
str1.zfill(10) = '0000Python'
```

Приклади роботи методів із рядками – str.find(), str.rfind(), str.index(), str.rindex():

```
str1 = 'Python th'
print(f'{str1 = }')
str2 = 'Language'
print(f'{str2 = }')
print(f'{str1.find("th") = }, {str2.find("th") = }')
print(f'{str1.index("th") = }')
print(f'{str1.rfind("th") = }, {str2.rfind("th") = }')
print(f'{str1.rindex("th") = }')
try:
    print(f'{str2.index("th")}')
except ValueError:
    print(f'There is no "th" in string {str2}')

Результат роботи коду:
str1 = 'Python th'
str2 = 'Language'
str1.find("th") = 2, str2.find("th") = -1
str1.index("th") = 2
str1.rfind("th") = 7, str2.rfind("th") = -1
str1.rindex("th") = 7
There is no "th" in string Language
```

Приклади роботи методів із рядками – str.split(), str.rsplit(), str.splitlines(), str.partition(), str.rpartition():

```
str1 = 'Python\tLanguage\t\n'
sep = "\t"
print(f'{str1.split(sep) = }')
print(f'{str1.rsplit(sep, maxsplit=1) = }')
print(f'{str1.splitlines() = }')
print(f'{str1.partition(sep) = }')
print(f'{str1.rpartition(sep) = }')

Результат роботи коду:
str1.split(sep) = ['Python', 'Language', '\n']
str1.rsplit(sep, maxsplit=1) = ['Python\tLanguage', '\n']
str1.splitlines() = ['Python\tLanguage\t']
str1.partition(sep) = ('Python', '\t', 'Language\t\n')
str1.rpartition(sep) = ('Python\tLanguage', '\t', '\n')
```

Приклади роботи методів із рядками – str.replace(), str.count(), str.encode(), str.join():

```
str1 = 'Python'
print(f'{str1 = }')
print(f'{str1.replace("on", "off") = }')
print(f'{str1.count("on") = }')
print(f'{str1.encode("ascii") = }')
print(f'{", ".join(str1) = }')

Результат роботи коду:
str1 = 'Python'
str1.replace("on", "off") = 'Pythoff'
str1.count("on") = 1
str1.encode("ascii") = b'Python'
", ".join(str1) = 'P, y, t, h, o, n'
```

Приклади роботи методів із рядками – str.format(), str.format_map():

```
str1 = 'Мова програмування - {language}'.format(language='Python')
print(f'{str1 = }')
map = {'language': 'Python', 'version': 3.11}
str2 = 'Мова програмування: {language}. Версія: {version}'
print(str2.format_map(map))

Результат роботи коду:
str1 = 'Мова програмування - Python'
Мова програмування: Python. Версія: 3.11
```

Приклади роботи методів із рядками – str.translate(), str.maketrans():

```
str1 = 'Python'
print(f'{str1 = }')
table = str.maketrans('Python', 'Пайтон')
str2 = str1.translate(table)
print(f'{str2 = }')

      Результат роботи коду:
```

```
str1 = 'Python'
str2 = 'Пайтон'
```

1.3.2. Тип даних списки (list)

Списки в python – це впорядковані колекції об'єктів довільних типів, які є змінними. Списки найбільш універсальний тип даних в python, які можуть включати такі основні вбудовані типи даних python:

- числові типи даних (numeric data types): цілі (integers), з рухомою комою (floating-point numbers); комплексні числа (complex);
- рядкові типи даних (string data types);
- логічні типи даних (boolean type): True і False;
- послідовності: списки (list), кортежі (tuple);
- словники (dictionary);
- набори (set);
- невизначене значення змінної (None).

Списки в python можна створювати кількома способами:

- Використання літерала – пари квадратних дужок для позначення порожнього списку: []
- Використання літерала – пари квадратних дужок, розділяючи елементи списку комами: [a], [a, b, c]
- Використання вбудованої в python функції: list(), якщо аргумент у функції list() не задано, створюється новий порожній список.
- Використання генераторів списків: [x for x in iterable].

Приклади коду створення списків:

```
print("Створення списків:")
print('за допомогою літерала:')
list0 = []
print(f'{list0 = }')
list1 = [3, 3.55, 'sp', ['ok', 1], {'a': 1, 'b': 2}]
print(f'{list1 = } \n')
print('за допомогою вбудованої функції list()')
str1 = '4564ab'
print(f'{str1 = }')
print(f'{list(str1) = } \n')

tuple1 = (45, 'a', 6.04, True)
print(f'{tuple1 = }')
print(f'{list(tuple1) = } \n')

dict1 = {1: True, 2: False}
print(f'{dict1 = }')
print(f'{list(dict1) = }\n')

print('за допомогою генераторів списків')
list5 = [i * 2 for i in 'abcdgc']
print(f'{list5 = }')
list6 = [a ** 3 for a in range(7)]
print(f'{list6 = } \n')
```

```
Результат роботи коду:
list0 =  []
list1 =  [3, 3.55, 'sp', ['ok', 1], (1, 2, 'a'), {'a': 1, 'b': 2},
{1, 2, 3}, None]
list4 =  ['4', '5', '6', '4', '4', '7']
list5 =  ['aa', 'bb', 'cc', 'dd', 'gg', 'cc']
list6 =  [0, 1, 4, 9, 16, 25, 36]
```

Необхідно дотримуватися особливостей синтаксису, характерних для кожного окремого типу даних в списку – лапки для рядків, числа і булеві значення без лапок, розділення елементів комами для списків, кортежів тощо.

Отримати доступ до елементів, збережених в списку можна, точно так, як і в рядках, за допомогою оператора нарізки ([] і [:]) і індексів, починаючи з нуля і до кінця списку.

Приклад коду використання оператора нарізки при роботі зі списками:

```
# Зрізи елементів списку
list1 = [76, 3.14, 'text', 70.2, (1, 'b')]
print(f'{list1 = }')
print(f'{list1[:] = }')
print(f'{list1[0] = }')
print(f'{list1[-1] = }')
print(f'{list1[:2] = }')
print(f'{list1[2:4] = }')
print(f'{list1[::2] = }')
print(f'{list1[0:3:2] = }')
```

```
Результат роботи коду:
list1 = [76, 3.14, 'text', 70.2, (1, 'b')]
list1[:] = [76, 3.14, 'text', 70.2, (1, 'b')]
list1[0] = 76
list1[-1] = (1, 'b')
list1[:2] = [76, 3.14]
list1[2:4] = ['text', 70.2]
list1[::2] = [76, 'text', (1, 'b')]
list1[0:3:2] = [76, 'text']
```

Розглянемо методи роботи зі списками:

list1.copy () – створює копію списку list1

list1.count (x) – повертає кількість елементів зі значенням x

list1.append (x) – додає елемент x в кінець списку

list1.extend(list2) – розширює список list1, додаючи в кінець всі елементи списку list2

list1.insert (i, x) – вставляє елемент у заданій позиції, перший аргумент – індекс елемента, перед яким потрібно зробити вставку, таким чином list1.insert(0, x) додає новий елемент на самому початку списку, а list1.insert(len(a), x) еквівалентно виразу list1.append(x).

list1.remove (x) – видаляє перший елемент у списку, який має значення x, при видаленні неіснуючого елемента повертається помилка.

list1.pop(i) – видаляє і повертає i-ий елемент, якщо індекс не вказано – list1.pop() видаляється останній елемент списку.

Видалити елемент зі списку через посилання на його індекс та частину списку можна і за допомогою оператора del: del list1[i].

list1.index(x, [start [, end]]) – повертає положення першого елемента зі значенням x у вказаному діапазоні від індексу start до індексу end, не включаючи останній індекс.

list1.sort(*, key=None, reverse=False)) – сортує список на основі функції, якщо аргументи відсутні, тобто list1.sort(), то функція сортує список у порядку зростання елементів списку.
list1.reverse () – формує список у зворотному порядку.
list1.clear () – очищає список та повертає пустий список.
Квадратні дужки ([]) *в аргументах методів означають, що аргументи взяті в квадратні дужки не є обов'язковими.*
* - *означає тип даних список*
** - *означає тип даних словник*

Приклад коду методів роботи зі списками – list.copy(), list.count():

```
# list.copy () - створює копію списку list1
list1 = [10, 20, 30, 50, 50, 10, 10, 0]
print(f'{list1 = }')
list2 = list1.copy()
print(f'list2 = list1.copy() {list2 = }')
# list.count(x) - повертає кількість елементів зі значенням x
print(f'{list1.count(50) = }  {list1.count(10) = } \n'
      f'{list1.count(0) = } {list1.count(66) = } \n')
```

```
Результат роботи коду:
list1 = [10, 20, 30, 50, 50, 10, 10, 0]
list2 = list1.copy() list2 = [10, 20, 30, 50, 50, 10, 10, 0]
list1.count(50) = 2  list1.count(10) = 3
list1.count(0) = 1 list1.count(66) = 0
```

Приклад коду методів роботи зі списками – list.append(x), list.extend(list2):

```
# append(x) - додає елемент x в кінець списку
list1 = ['Ivanov']
print(f'{list1 = }')
list1.append('Tronko')
print(f"list1.append('Tronko') \n {list1 = }")
# додати елемент в кінець списку можна,
# використавши оператор нарізки і функцію len(list2)
list1[len(list1):] = ['Petrov']
print(f"list1[len(list1):] = ['Petrov'] \n {list1 = }")
print('в append() не можна використовувати змінної,')
print('так як отримаємо list11 = None')
list11 = list1.append('Zubenko')
print(f"list11 = list1.append('Zubenko') \n {list11 = }")
# list1.extend(list2) - розширює список list1,
# додаючи в кінець всі елементи списку list2
list1 = ['Ivanov', 'Tronko', 'Petrov']
print(f'{list1 = }')
list2 = ['Sorokin', 'Sorokina']
print(f'{list2 = }')
list1.extend(list2)
print(f'list1.extend(list2) \n {list1 = } \n')
```

```
      Результат роботи коду:
list1 = ['Ivanov']
```

```
list1.append('Tronko')
 list1 = ['Ivanov', 'Tronko']
list1[len(list1):] = ['Petrov']
 list1 = ['Ivanov', 'Tronko', 'Petrov']
в append() не можна використовувати змінної,
так як отримаємо list11 = None
list11 = list1.append('Zubenko')
 list11 = None
Методи роботи зі списками list.extend(list2):
list1 = ['Ivanov', 'Tronko', 'Petrov']
list2 = ['Sorokin', 'Sorokina']
list1.extend(list2)
 list1 = ['Ivanov', 'Tronko', 'Petrov', 'Sorokin', 'Sorokina']
```

Приклад коду методів роботи зі списками - list.insert(), list.remove(), list.pop():

```
list1 = [10, 20, 'ff', 30, 'ff', 30, 50, 50, 0]
print(f'list1 = {list1}')
# функція list1.insert(i, x) вставляє у список list елемент x в позицію з індексом i
list1.insert(3, [1, 2, 3])
print(f'list1.insert(3, [1, 2, 3]) \n {list1 = }')
# list1.remove (x) видаляє з list елемент x, який зустрічається першим
list1.remove('ff')
print(f"list1.remove('ff') \n {list1 = }")
list1 = [10, 20, 'ff', 30, 'ff', 30, 50, 50, 0]
print(f'{list1 = }')
# функція list.pop(i) видаляє зі списку елемент з індексом i
list1.pop(4)
print(f'list1.pop(4)  {list1 = }')
# якщо аргуменрт у функції list.pop() не задано, видаляється зі списку останній елемент
list1.pop()
print(f'list1.pop() {list1 = }')
print('Видалення елементів зі списку за допомогою оператора del')
list1 = [10, 20, 'ff', 30, 'ff', 30, 50, 50, 0]
print(f'{list1 = }')
del list1[0]
print(f'del list1[0]  {list1 = }')
#  del  list1[1:3]  -  видалення  підпослідовності  зі  списку  з  індексами від і до j,
#не включаючи елемент з індексом j (використання зрізів)
del list1[1:3]
print(f'del list1[1:3]  {list1 = } \n')
Результат роботи коду:
list1 = [10, 20, 'ff', 30, 'ff', 30, 50, 50, 0]
list1.insert(3, [1, 2, 3])
 list1 = [10, 20, 'ff', [1, 2, 3], 30, 'ff', 30, 50, 50, 0]
list1.remove('ff')
list1 = [10, 20, [1, 2, 3], 30, 'ff', 30, 50, 50, 0]
list1 = [10, 20, 'ff', 30, 'ff', 30, 50, 50, 0]
list1.pop(4)  list1 = [10, 20, 'ff', 30, 30, 50, 50, 0]
```

```
list1.pop() list1 = [10, 20, 'ff', 30, 30, 50, 50]
Видалення елементів зі списку за допомогою оператора del
list1 = [10, 20, 'ff', 30, 'ff', 30, 50, 50, 0]
del list1[0]  list1 = [20, 'ff', 30, 'ff', 30, 50, 50, 0]
del list1[1:3]  list1 = [20, 'ff', 30, 50, 50, 0]
```

Приклади методів роботи зі списками: list.index(x, [start [, end]]), list.sort(), list.reverse(), list.clear():

```
# list.index(x, [start [, end]]) - повертає положення першого
елемента зі #значенням x  у вказаному діапазоні від індексу start
до індексу end, не #включаючи останній індекс
list1 = [10, 20, 50, 10, 0]
print(f'{list1 = }')
print(f'{list1.index(10) = }')
print(f'{list1.index(10, 1, 4) = } \n')

#list.sort(*, key=None, reverse=False)) - сортує список на основі
функції, якщо #аргументи відсутні, то функція сортує список у
порядку зростання #елементів списку')
list1 = [10, 20, 30, 30, 50, 50, 0]
print(f'{list1 = }')
list1.sort()
print(f'list1.sort()  = {list1}')
# Аналогічно працює вбудована функція python sorted() в циклі for
for i in sorted(list1):
    print(f'{i}')
print(f'{list1 = }')
list1.reverse()
print(f'list1.reverse(): list1 = }')
# для функції list.reverse() не можна використовувати змінної,
# так як отримаємо list2 = None
list2 = list1.reverse()
print(f'list2 = list2.reverse() {list2 = } \n')
#list1.clear() - очищає список та повертає пустий список
list1.clear()
print(f'{list1 = }')
list1 = [10, 20, 30, 10, 0]
print(f'{list1.clear() = } \n')
```

```
Результат роботи коду:
list1 = [10, 20, 50, 10, 0]
list1.index(10) = 0
list1.index(10, 1, 4) = 3

list1 = [10, 20, 30, 30, 50, 50, 0]
list1.sort()  = [0, 10, 20, 30, 30, 50, 50]
0
10
20
30
30
50
```

```
50
list1 = [0, 10, 20, 30, 30, 50, 50]
list1.reverse(): list1 = [50, 50, 30, 30, 20, 10, 0]
list2 = list2.reverse() list2 = None
list1 = []
list1.clear() = None
```

1.3.3. Тип даних кортежі (tuple)

Кортежі дуже схожі на списки, але мають одну важливу відмінність – вони незмінні. В іншому, вони також можуть складатися з даних різних типів як і списки і використовувати індекси, які визначають конкретний порядок елементів. Індекс починається з нуля, як і в разі списків, а негативний індекс з -1, який вказує на останній елемент кортежу.

Створення кортежів довжиною в один елемент представляє певну проблему, але для цього існують синтаксичні особливості. Пусті кортежі утворюються як і списки за допомогою пустої пари дужок, а одноелементний кортеж утворюється за допомогою певного елемента та коми.

Аналогічно спискам кортежі в python можна створювати кількома способами:

- Використання літерала – пари круглих дужок для позначення порожнього кортежу: ()
- Використання літерала – пари круглих дужок, розділяючи елементи кортежу комами: (a,), (a, b, c)
- Використання вбудованої в python функції: tuple(), якщо аргумент функції tuple() не задано, створюється новий порожній кортеж.

Можна використовувати кортежі замість списків, але вони мають менше можливостей, так як не можуть бути змінені після створення.

Кортежі мають багато застосувань. Наприклад: пара координат (x, y), аргументи функцій, запис із бази даних тощо. Кортежі, подібно до рядків, є незмінними: присвоєння нового значення певному елементу кортежу – неможливе (втім, це можна зробити за допомогою зрізів та конкатенації. Можливо також створити кортежі, що містять змінні об'єкти, такі як списки.

Переваги кортежів:

- використання кортежів в задачах, де потрібно забезпечити підтримку цілісності, тобто кортеж не може бути змінений з іншого посилання;
- використання кортежів в якості ключів до елементів словників (ключі словника не можна змінювати);
- кортежі необхідні, коли потрібно використовувати фіксовані набори об’єктів;
- аргументи функцій можна передавати як кортежі;
- кортежі займають менше місця.

Приклади коду створення кортежів:

```
print('Створюємо кортеж за допомогою:')
print(' - літерала')
tuple0 = ()
print(f'{tuple0 = }')
tuple1 = (23,)
print(f'{tuple1 = }')
tuple11 = (23)
print(f'одноелементний кортеж без коми \n'
      f'створити не можливо \n {tuple11 = }'
tuple2 = ('1', 3.7, (1, 2, 20), True, {'a': 11, 'b': 12})
print(f'{tuple2 = }')
```

```
print(' - вбудованої функції tuple():')
tuple3 = tuple('hello, world!')
print(f'- з рядка: \n {tuple3 = }')
tuple4 = tuple([1, 's', 5.5, None])
print(f'- зі списку: \n {tuple4 = }')
tuple5 = tuple({'a': 11, 'b': 12})
print(f'- зі словника (ключі): \n {tuple5 = }')
Результат роботи коду:
Створюємо кортеж за допомогою:
 - літерала
tuple0 = ()
tuple1 = (23,)
одноелементний кортеж без коми створити не можливо
 tuple11 = 23
tuple2 = ('1', 3.7, (1, 2, 20), True, {'a': 11, 'b': 12})
 - вбудованої функції tuple():
- з рядка:
 tuple3 = ('h', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l',
'd', '!')
- зі списку:
 tuple4 = (1, 's', 5.5, None)
- зі словника (ключі):
 tuple5 = ('a', 'b')
```

Так як кортежі є незмінним типом даних, кількість методів роботи з ними значно менша ніж зі списками.

Розглянемо методи роботи з кортежами:

tuple1.index() - використовується для виведення індексу елементу.

tuple1.count() - використовується для підрахунку кількості входження елементу в кортеж.

len() - показує кількість елементів кортежу, використовуються для всіх типів даних.

Нижчеперелічені методи (функції) python використовуються для кортежів і для всіх типів даних, які складаються тільки з числових типів даних.

sum() – розраховує суму всіх елементів кортежу.

min() - показує елемент кортежу з найменшим значенням.

max() - показує елемент кортежу з максимальним значенням.

При виведенні на друк кортежі завжди оточуються дужками, що допомагає правильно інтерпретувати вкладені кортежі. При вводі вони також можуть оточуватися дужками, хоча і не обов'язково. У більшості ж випадків дужки потрібні, якщо кортеж є частиною більшого виразу.

Приклади методів роботи з кортежами: tuple.count (), tuple.index():

```
a = 10
tuple1 = (10, 'St', 'a', a**2, 'St', (1, 2, 20), [1, 9, 'ff'],
'St')
print(f'tuple1 = {tuple1}')
# count(x) повертає кількість елементів x в кортежі
n = tuple1.count('St')
print(f"Number of string 'St' in the tuple1: {n = } ")
n = tuple1.count(10)
print(f"Number of string 'St' in the tuple1: {n = } \n")
# функція index(x) повертає індекс елементу x в кортежі
ind = tuple1.index('St')
```

```
ind1 = tuple1.index([1, 9, 'ff'])
print(f"index елементу 'st': {ind = } \n [1, 9, 'ff']: {ind = }
\n")
  Результат роботи коду:
tuple1 = (10, 'St', 'a', 100, 'St', (1, 2, 20), [1, 9, 'ff'],
'St')
Number of string 'St' in the tuple1: n = 3
Number of 10 in the tuple1: n1 = 1
index елементу 'St': ind = 1
 [1, 9, 'ff']: ind1 = 6
```

1.3.4. Тип даних словники (dictionary)

Словники в python – невпорядковані набори пар ключ : значення, де ключі не повинні повторюватися в межах одного словника. Словники можна зустріти в інших мовах програмування під назвою "асоціативна пам'ять" чи "асоціативні масиви".

На відміну від послідовностей, що індексуються за допомогою чисел, словники індексуються за допомогою ключів, які можуть належати до будь-якого незмінного типу, тобто ключі можуть задаватися типами даних рядки і числа. Кортежі також можуть бути ключами словника, якщо вони складаються лише з рядків, чисел чи кортежів. Якщо ж кортеж містить, прямо чи опосередковано, певний змінний об'єкт, то він не може використовуватися як ключ. Використання списків у якості ключів неможливе, тому що списки можуть змінюватися методи append() та extend(), а також шляхом присвоєння через зрізи та індексацію.

Словники в python можна створювати кількома способами:

- Використання літерала – пари фігурних дужок для позначення порожнього словнику: {}
- Використання літерала – пари фігурних дужок, розділяючи елементи ключ : значення словнику комами: [‘a’ : 1], [‘a’ : 1, ‘b’ : 2, ‘c’ : 3]
- Використання вбудованої в python функції dict(), яка створює словники зі списків пар ключ : значення, що задані кортежами, якщо аргумент у функції dict() не задано, створюється новий порожній словник
- Використання генераторів словників, які дуже схожі на генератори списків: {x for x in iterable}.

Приклади коду створення словників:

```
print('- за допомогою літерала')
d0 = {}
print(f'{d0 = }')
dict1 = {'Ivanov': 90, 'Petrov': 65, 'Petrova': 75}
print(f'{dict1 = } \n')

print('- за допомогою функції dict()')
dict2 = dict([('a', 1), ('b', 2), ('c', 3)])
print(f'{dict2 = }')
dict3 = dict(short='dict', long='dictionary')
print(f'{dict3 = } \n')

print('- за допомогою генератору словника')
dict4 = {a: a ** 2 for a in range(7)}
print(f'{dict4 = } \n')
  Результат роботи коду:
- за допомогою літерала
d0 = {}
```

```
dict1 = {'Ivanov': 90, 'Petrov': 65, 'Petrova': 75}

- за допомогою функції dict()
dict2 = {'a': 1, 'b': 2, 'c': 3}
dict3 = {'short': 'dict', 'long': 'dictionary'}

- за допомогою генератору словника
dict4 = {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

Основні операції зі словником – збереження значення за певним ключем і отримання значення для даного ключа. Якщо Ви зберігаєте значення для ключа, що вже існує, то старе значення, прив'язане до ключа не зберігається. Спроба отримання значення для неіснуючого ключа призводить до помилки.

Розглянемо методи роботи зі словниками:

dict.items() – повертає пари (ключ, значення).

dict.popitem() – видаляє першу пару ключ : значення у словнику, якщо словник порожній, повертає помилку KeyError 'popitem(): dictionary is empty'

dict.pop(key [, default]) – видаляє ключ і повертає значення по заданому ключу, якщо ключ не вказано або вказаного ключа немає, повертає помилку KeyError:

dict.keys() – повертає список всіх ключів у словнику

dict.values() – повертає список всіх значень у словнику

dict.get(key [, default]) – повертає значення по заданому ключу, якщо ключ відсутній, повертає помилку TypeError: get expected at least 1 argument, got 0

dict.update ([other]) – оновлює словник, додаючи пари (ключ, значення)

dict.copy() – повертає копію словника, працює аналогічно цій функціє для списку

dict.clear() – очищає словник, видалення пари ключ : значення можливе за допомогою оператора del; функція clear() і оператора del працюють аналогічно цим операціям для списку

Приклад коду методу роботи зі словниками - dict.items():

```
# Створюємо словник, в якому ім'я – це ключ, бали – це значення
dict1 = {'Ivanov': 90, 'Petrov': 65}
print(f'{dict1 = } {type(dict1) = }')
while True:
    name = input('What is your name?\n')
    if name == '':
        break
    score = input('What is your score? \n')
    if score == '':
        break
    dict1[name] = score
# Метод items() повертає ключі та значення словника
# змінна k зберігає ключ, змінна v зберігає значення
for k, v in dict1.items():
    print(k, ':', v)
print()
Результат роботи коду (варіант коду, коли name == ''):
dict1 = {'Ivanov': 90, 'Petrov': 65}
What is your name?

{'Ivanov': 90, 'Petrov': 65}
Ivanov : 90
Petrov : 65
```

```
Результат роботи коду (варіант коду, коли name == ' Ivanov ', а
score  == 100):
dict1 =  {'Ivanov': 90, 'Petrov': 65}
What is your name?
Ivanov
What is your score?
100
What is your name?

Ivanov : 100
Petrov : 65
```

Приклади методів роботи зі словниками: copy(), update(), keys(), values():

```
dict1 = {'Ivanov': 90, 'Petrov': 65}
print(f'{dict1 = }')
dict2 = dict1.copy()
print(f'dict2 = dict1.copy() \n {dict2 = } \n')
#dict.update() оновлює словник, додаючи пари (ключ, значення)')
dict1.update({'Sidorov': 75, 'Ivanko': 99})
print(f"dict1.update('Sidorov': 75, 'Ivanko': 99) \n {dict1 = }")
#в dict.update([other]) не можна використовувати змінної')
dict11 = dict1.update({'Sidorov': 75, 'Ivanko': 99})
print(f"dict11 = dict1.update('Sidorov': 75, 'Ivanko': 99) \n
{dict11 = }")
#dict.keys() повертає ключі, values() - список значень словника')
print(f'{dict1.keys() = }')
print(f'{dict1.values() = } \n')
  Результат роботи коду:
dict1 = {'Ivanov': 90, 'Petrov': 65}
dict2 = dict1.copy()
 dict2 = {'Ivanov': 90, 'Petrov': 65}
dict1.update('Sidorov': 75, 'Ivanko': 99)
 dict1 = {'Ivanov': 90, 'Petrov': 65, 'Sidorov': 75, 'Ivanko': 99}
dict11 = dict1.update('Sidorov': 75, 'Ivanko': 99)
 dict11 = None
dict1.keys() = dict_keys(['Ivanov', 'Petrov', 'Sidorov',
'Ivanko'])
dict1.values() = dict_values([90, 65, 75, 99])
  Приклади методів роботи зі словниками - get(key [, default],
clear():
dict1 = {'Ivanov': 90, 'Petrov': 65, 'Butko': 85, 'Prutko': 45}
print(f'{dict1 = }')
# dict.get(key [, default]) - повертає значення по заданому ключу,
# якщо ключ відсутній, повертає default (за замовчуванням None)
get_dict1 = dict1.get('Butko')
print(f"get_dict1 = dict1.get('Butko'): {get_dict1} \n")
dict1.clear()
print(f' dict1.clear()  {dict1 = }')
dict1 = {'Ivanov': 90, 'Petrov': 65, 'Butko': 85, 'Prutko': 45}
print(f'{dict1.clear() = } \n')
  Результат роботи коду:
```

```
dict1 = {'Ivanov': 90, 'Petrov': 65, 'Butko': 85, 'Prutko': 45}
get_dict1 = dict1.get('Butko'): 85
dict1.clear()  dict1 = {}
dict1.clear() = None
```

1.3.5. Тип даних множини (set)

Множина (set) в python – це невпорядкована сукупність неповторюваних елементів.

Серед основних застосувань множин є такі як перевірка приналежності (членства) тих чи інших об’єктів до множини, визначення чи знаходиться один набір всередині іншого, знаходження перетин між двома наборами тощо. Множини використовуються тоді, коли існування об’єктів в колекції чи те скільки разів вони зустрічаються, є більш важливим ніж порядок їх розташування. З об'єктами, що виражають множини, здійснюються такі математичні операції як об'єднання (union), перетин (intersection), різниця (difference), та симетрична різниця (symmetric difference).

Як і списки, множини дозволяють внесення і видалення елементів. Множини можуть включати різні типи даних, але множини не підтримують змінювані елементи, такі як списки та словники.

Множини можуть включати такі вбудовані типи даних python:

- всі числові типи даних;
- рядкові типи даних;
- послідовності: кортежі;
- невизначене значення змінної (None).

Основні особливості множин:

- множини не містить елементів, що повторюються;
- елементи множини розташовуються в довільному порядку, тобто не індексуються, тому набори не підтримують ніяких операцій зрізу і індексування.
- множини в python можна створювати кількома способами:
- використання літерала – пари фігурних дужок, розділяючи елементи множини комами: {‘a’, ‘b’, 2, ‘c’, 3}. За допомогою літерала {} створити пусту множину неможливо, тільки словник.
- використання вбудованої в python функції set(), за допомогою якої можна створити пусту множину; множину з об’єктів типу числа, рядки, кортежі; множину зі списку або кортежу.
- використання генераторів множин: {x for x in iterable}, які аналогічні генераторам списків і кортежів.

Приклади коду створення множин:

```
print('- за допомогою літерала')
print('за допомогою літерала {} створити пусту множину неможливо,'
'тільки словник')
set0 = {}
print(f'{set0 = }, {type(set0) = }')
set1 = {1, 1, 2.3, (2.3 - 4.6j), 'Hi', True, ('g', 1, 0, 'p'),
None}
print(f"{set1 = } {type(set1)}\n")

print('- за допомогою вбудованої функції set()')
set2 = set()
print(f'пусту множину: {set2 = } {type(set2)}\n')

list_fruits = ['apple', 'plum', 'apple', 'orange', 'pear', 'plum',
```

```
'banana']
set21 = set(list_fruits)
print(f'set() створює множину set21 зі списку list_fruits: \n
{set21 = }')

tuple_fruits = ('apple', 'orange', 'apple', 'orange', 'pear',
'orange', 'banana')
set22 = set(tuple_fruits)
print(f'set() створює множину set21 з кортежу tuple_fruits: \n
{set22 = }')

set3 = set([1, 'Hi', ('g', 1, 0, 'p'), 'Hi'])
print(f'set() створює множину set3 зі списку, який є аргументом
функції' 'set(): \n {set3 = }\n')

set4 = set((1, 'Hi', ('g', 1, 0, 'p'), 'Hi'))
print(f'set() створює множину set4 з кортежу, який є аргументом
функції' 'set(): \n {set4 = }\n')
# Виклик функції можна замінити літералом
set31 = {1, 'Hi', ('g', 1, 0, 'p'), 'Hi'}
set5 = {a ** 3 for a in range(7)}
print(f'- за допомогою генератору множин: \n {set5 = } \n')
```

```
  Результат роботи коду:
- за допомогою літерала
за допомогою літерала {} створити пусту множину неможливо, тільки
словник
set0 = {}, type(set0) = <class 'dict'>
set1 = {1, 2.3, ('g', 1, 0, 'p'), (2.3-4.6j), 'Hi', None} <class
'set'>
- за допомогою вбудованої функції set()
пусту множмну: set2 = set() <class 'set'>
set() створює множину set21 зі списку list_fruits:
 set21 = {'banana', 'apple', 'pear', 'orange', 'plum'}
set() створює множину set21 з кортежу tuple_fruits:
 set22 = {'pear', 'banana', 'orange', 'apple'}
set() створює множину set3 зі списку, який є аргументом функції
set():
 set3 = {1, ('g', 1, 0, 'p'), 'Hi'}
set() створює множину set4 з кортежу, який є аргументом функції
set():
 set4 = {1, ('g', 1, 0, 'p'), 'Hi'}
- за допомогою генератору множин:
 set5 = {0, 1, 64, 8, 216, 27, 125}
```

Розглянемо методи роботи з множинами:

set.add(x) - додає елемент до набору

update([]) - додає список до набору

x in set1 – перевіряє приналежність елементу x до множини set1

set == set1- перевіряє чи всі елементи множини set належать set1, чи всі елементи множини set1 належать множині set

set.issubset (set1) або set <= set1 - перевіряє чи всі елементи set належать set1

set.union(set1, set2,...) або set | set1 | set2... - об'єднує декількох множин

set.intersection(set1, set2,...) або set & & ... - …перетин декількох множин
set.isdisjoint(set1) – повертає True, якщо set і set1 не мають спільних елементів.
set.remove(x) - видаляє елемент x з множини, повертає помилку, якщо такого елемента не існує: KeyError
set.discard(x) - видаляє елемент x, якщо він знаходиться у множині (discard - відкинути).
set.pop() - видаляє перший елемент з множини, так як набори не впорядковані, не можна точно сказати, який елемент буде першим
set.clear() - очищення множину
len(set1) – повертає число елементів у множині.

Приклади методів роботи з множинами: set.add(),set.update():

```
set1 = {1, 'Hi', ('g', 1, 0, 'p'), 43.2}
print(f'{set1 = } \n')
# set.add() додає один елемент до множини')
set1.add('++k')
print(f"set1.add('++k'): \n {set1 = } \n")
# set.update() додає список до множини')
set1.update(['j', ('1', 1, 1, 1)])
print(f'set1.update(): \n {set1 = } \n')
  Результат роботи коду:
set1 = {1, 43.2, ('g', 1, 0, 'p'), 'Hi'}
set1.add('++k'):
 set1 = {1, 43.2, 'Hi', ('g', 1, 0, 'p'), '++k'}
set1.update():
 set1 = {1, 43.2, ('1', 1, 1, 1), 'Hi', ('g', 1, 0, 'p'), 'j',
'++k'}
```

Приклади методів роботи з множинами – set.issubset(set1) або set <= set1, set.union():

```
set1 = {1, 2, 3, 4, 5, 6, 7}
set2 = {2, 0, -1, 4, 7, -10}
set3 = {1, 2, 4, 3, 5, 6, 7}
set4 = {1, 2, 3, 4, 5, 6, 7, 8, 9}
#set1.issubset(set2) та set1 <= set2 перевіряють, \n
#чи всі елементи set1 включені в set2, повертають True або False
print(f'set1.issubset(set2): {set1.issubset(set2)},
set1.issubset(set3): {set1.issubset(set3)},\n'
      f'set1.issubset(set4): {set1.issubset(set4)},
set4.issubset(set1): {set4.issubset(set1)}\n')

print('set1 <= set2', set1 <= set2, 'set1 <= set3', set1 <= set3,
      'set1 <= set4', set1 <= set4, 'set4 <= set1', set4 <= set1,
'\n')
#set.union() або set | set1 | set2... об'єднує декілька множин
print(f'set.union(set1, set2, set3, set4): {set.union(set1, set2,
set3, set4)} \n')
print(f'set1 | set2 | set3 : {set1 | set2 | set3} \n')
  Результат роботи коду:
set1.issubset(set2): False, set1.issubset(set3): True,
set1.issubset(set4): True, set4.issubset(set1): False
set1 <= set2 False set1 <= set3 True set1 <= set4 True set4 <=
set1 False
set.union(set1, set2, set3, set4): {0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
-10, -1}
```

Приклади методів роботи з множинами: x in set1, set1 == set2')

```
set1 = {1, 2, 3, 4, 5, 6, 7}
set2 = {2, 0, -1, 4, 7, -10}
set3 = {1, 2, 4, 3, 5, 6, 7}
set4 = {1, 2, 3, 4, 5, 6, 7, 8, 9}
#метод x in set1 перевіряє приналежність x до set1, повертає True
або False
print(f'4 in set1: {4 in set1},  8 in set1: {8 in set1} \n')

#set1 == set2 перевіряє, чи всі елементи множини set1 належать
set2, \n'
#чи всі елементи множини set2 належать множині set1, повертає True
або False
print('set1 == set2', set1 == set2, 'set1 == set3', set1 == set3,
'set2 == set3', set2 == set3, 'set3 == set4', set3 == set4, '\n')
Результат роботи коду:
4 in set1: True,  8 in set1: False
set1 == set2 False set1 == set3 True set2 == set3 False set3 ==
set4 False
```

Приклади методів роботи з множинами – set.intersection(set1, set2,...) або set1 & set2 & ... перетин (intersection):

```
#методи set.intersection(set1, set2,...) або set1 & set2 & ...
перетин
#(intersection)декількох множин, тобто повертає елементи, які
наявні у всіх #вказаних множинах
set1 = {1, 'Hi', ('g', 1, 0, 'p'), 43.2}
set2 = {2, 0, 1, 4, 7, -10, 43.2, 'Hi'}
set3 = {None, (2.3 - 4.6j), 7}
print(f'set.intersection(set1, set2): {set.intersection(set1,
set2) = }')
print(f'set1 & set2: {set1 & set2 = } \n')
# set1.isdisjoint(set2) – повертає True, якщо set і set1 не мають
спільних #елементів, disjoint – непересічний, інакше – False')
print(f'set1.isdisjoint(set2): {set1.isdisjoint(set2)} ')
print(f'set1.isdisjoint(set3): {set1.isdisjoint(set3)} \n')
  Результат роботи коду:
set.intersection(set1, set2): set.intersection(set1, set2) = {1,
43.2, 'Hi'}
set1 & set2: set1 & set2 = {1, 43.2, 'Hi'}
set1.isdisjoint(set2): False
set1.isdisjoint(set3): True
```

Приклади методів роботи з множинами - set1 ^ set2 ^ se32, (^xor); set1.difference(set2, set3);

```
# метод set1 ^ set2 ^ set3 повертає симетричну різницю, тобто
повертає
# різницю між (set1 ^ set2), потім різницю між результатом (set1 ^
set2)
# та set3 і т.д., виключаючи співпадаючі елементи')
set1 = {1, 'Hi', ('g', 1, 0, 'p'), 43.2}
set2 = {2, 0, 1, 4,'Hi', 7, -10, 43.2, 'Hi'}
```

```
set3 = {None, 'Hi', (2.3 - 4.6j), 7, 43.2}
print(f'set1 ^ set2: {set1 ^ set2}')
print(f'set1 ^ set2 ^ set3: {set1 ^ set2 ^ set3}')
print(f'set1 ^ set3: {set1 ^ set3}')
#метод set1.difference(set2, set3) повертає унікальні елементи
# для конкретної множини set1, яких немає в інших множинах')
print('set1.difference(set2, set3)  ', set1.difference(set2,
set3))
print('set2.difference(set1, set3)  ', set2.difference(set1,
set3))
print('set3.difference(set1, set2)  ', set3.difference(set1,
set2), '\n')
```

Результат роботи коду:

```
set1 ^ set2: {0, 2, 4, 7, ('g', 1, 0, 'p'), -10}
set1 ^ set2 ^ set3: {0, 2, 4, ('g', 1, 0, 'p'), None, 43.2, 'Hi',
(2.3-4.6j), -10}
set1 ^ set3: {1, (2.3-4.6j), 7, ('g', 1, 0, 'p'), None}
set1.difference(set2, set3)   {('g', 1, 0, 'p')}
set2.difference(set1, set3)   {0, 2, 4, -10}
set3.difference(set1, set2)   {None, (2.3-4.6j)}
```

Приклади методів роботи з множинами – set.discard(x) та set.remove(x):

```
set1 = {('1', 1, 'Hi'), 1, 'k', ('g', 1, 0, 'k'), 'j', '1', 'Hi'}
print(f'{set1 = }')
# метод discard(x) видаляє елемент x з набору
set1.discard('Hi')
print("set1.discard('Hi') = ", set1)
# метод remove(x) видаляє елемент x з набору
set1.remove('k')
print("set1.remove('k') = ", set1, '\n')
```

Результат роботи коду:

```
set1 = {1, 'j', 'Hi', '1', ('g', 1, 0, 'k'), ('1', 1, 'Hi'), 'k'}
set1.discard('Hi') =  {1, 'j', '1', ('g', 1, 0, 'k'), ('1', 1,
'Hi'), 'k'}
set1.remove('k') =  {1, 'j', '1', ('g', 1, 0, 'k'), ('1', 1,
'Hi')}
```

Приклади методів роботи з наборами set.pop(),set.clear():

```
set1 = {1, 2, 3, 4, 5, 6, 7}
# pop() - видаляє перший елемент з множини, так множини не
впорядковані,
# видаляє випадковий елемент
print(f'set1.pop(): {set1.pop()}')
# set.clear() - очищення множину
set1.clear()
print(f'set1.clear(): {set1 = }')
print(f'set1.clear(): {set1.clear()}')
```

Результат роботи коду:

```
set1.pop(): 1
set1.clear(): set1 = set()
set1.clear(): None
```

Завдання та запитання до підрозділу «Складні типи даних python – рядки, списки, кортежі, словники, множини»

1. Виберіть варіанти, в яких правильно задано рядок (тип даних):

```
a = 'nbvlhl'
a = "nbvlhl12"
a = """nbvlhl"""
a = "nb'nbnb'lhl"
a = """nbv"nbnb"lhl"""
#a =  " "nbvlhl" "
#a =  ""nbvlhl""
#a = ""nbv"nbnb"lhl""
#a = ' 'nbvlhl' '
```

2. Який символ (символи) пригнічує екранування рядка?
3. Виберіть рядки коду, де друкується зріз всього рядка:

```
print(str1[:])
print(str1[::])
print(str1[1:])
print(str1[:-1])
print(str1[0:-1]).
```

4. Виберіть рядки коду, де друкується зріз елементів всього рядка у зворотному порядку:

```
print(str1[::-1])
print(str1[-1])
print(str1[:-1]).
```

5. Виберіть методи, які використовуються для роботи з кортежами: .index(), .count(), .isalnum(), .extend(), .pop().
6. Виберіть методи, які використовуються для роботи з рядками: .split(), .isdigit(), .isalnum(), .extend(x), .sort (), .insert (i, x).
7. Виберіть методи, які використовуються для роботи зі списками: .copy (), .append(x), .extend (L), islower(), .isupper().
8. Виберіть методи, які використовуються для роботи з наборами (множинами): .remove(x), .issubset (s), .add(x), .isalnum(), .items().
9. Виберіть методи, які використовуються для роботи зі словниками: .get(), .items(), .pop(), .values (), .keys(), .union(d1, d2), .append (x), .isalpha().
10. Які типи даних є незмінними?

1.4. Робота з файлами

1.4.1. Загальний огляд функції open()

У програмуванні на python робота з файлами є поширеною практикою, оскільки часто для обробки даних потрібно зчитувати їх зі зовнішніх файлів. Це дозволяє ефективно взаємодіяти з великим обсягом інформації, зберігати конфігураційні дані, а також забезпечує можливість зберігання даних між різними сеансами роботи програми.

Загалом файлом називають іменовану локацію на фізичному диску, де дані постійно зберігаються для можливості надалі отримати до них доступ. Файли можна розділити на два типи: бінарні і текстові. Оскільки комп'ютери завжди зберігають файли у бінарній формі – як набір з нулів та одиниць. Тобто файли першочергово представлені серіями бітів, записаних один за одним. У бінарній формі людина не може прочитати вміст файлу, тому для цього краще підходять текстові файли, які складаються із символів та можуть бути відкриті чи змінені за допомогою текстових редакторів.

До бінарних файлів можна віднести такі формати як .jpeg, .png (зображення), .mp3, .aac (звукові файли), .mp4, .mov, .mkv (відео), .exe, .msi (виконувані файли) тощо. При спробі відкрити ці файли за допомогою текстових редакторів можна буде спостерігати лише набір

символів, що не формують текст. Варто також пам’ятати, що при заміні навіть одного біту файл можливо зламати без можливості відновлення.

До текстових файлів відносяться такі відомі багатьом формати як .txt, .csv, .html, .json, а також .py – формат файлів python. При відкритті цих файлів у текстових редакторах можна побачити знайомі нам символи – літери. цифри та спеціальні символи. Проте на диску ці дані також зберігаються у бінарній формі, а їх перетворення у символи відбувається завдяки використанню спеціального кодування, наприклад, ASCII чи UNICODE.

У мові програмування робота з основними типами файлів не вимагає додаткових пакетів. Для доступу до вмісту файлу використовується вбудована функція open(), що має два основних аргументи: шлях до файлу та режим роботи з файлом [13].

Шлях до файлу може бути абсолютним або відносним. Абсолютним шляхом називають повний шлях від кореневого диску (або директорії для Linux-подібних операційних систем). Відносний шлях є коротшим і вказує на потрібний файли відносно того, в якій директорії розмішується виконуваний файл. Наприклад, шлях "C:\Users\user\Desktop\test.txt" є абсолютним, оскільки починається із назви диску (диск C), а шлях "test.txt" є відносним для папки Desktop та всіх файлів у ній.

При вказанні шляху до файлу варто пам’ятати про існування екранованих послідовностей (наприклад, '\n', '\t'), які можуть змінити рядок. Тому варто використовувати символ '/' або вказувати літеру r чи R перед рядком шляху. Наприклад, r'C:\Users\admin\Desktop\test.txt'.

Режим роботи з файлом за замовчуванням встановлено на читання, але різних режимів існує багато. Вони позначаються як рядок: позиційно другим аргументом або через параметр mode. Серед основних режимів можна виділити такі:

- 'r' – читання; якщо файл не існує, то виникне помилка FileNotFoundError: [Errno 2] No such file or directory,
- 'w' – запис із видаленням попереднього вмісту файлу; якщо файл не існує, то він буде створений без виникнення помилки,
- 'x' – запис у новий файл; якщо файл вже існує, то виникне помилка FileExistsError: [Errno 17] File exists,
- 'a' – запис із додаванням даних у кінець файлу та без видаленням попереднього вмісту; якщо файл не існує, то він буде створений без виникнення помилки,
- 'b' – робота з файлом у бінарній формі; може використовуватись лише разом з іншими режимами (наприклад, 'rb' чи 'ab'),
- 't' – робота з файлом у текстовій формі; використовується за замовчуванням та лише з іншими режимами (наприклад, 'rt' чи 'at'),
- '+' – дозволяє одночасно читати та записувати у файл; може використовуватись лише з іншими режимами (наприклад, 'r+' чи 'w+').

Варто розуміти, що режими 'r+' та 'r+b' створюють помилку за відсутності файлу, а 'a+', 'a+b', 'w+' та 'w+b' – ні. Режими 'x+' та 'x+b' навпаки викликають помилку, якщо файл вже існує. Комбінування режимів дозволяє не лише записувати та читати файли у різних формах, але й фільтрувати чи існують файли з вказаним ім’я.

Після розгляду основних параметрів функції open() варто зазначити, що її результатом є об’єкт класу _io.TextIOWrapper, що має свої методи та параметри. Метод close() використовується для закриття відкритого файлу, що дозволяє звільнити ресурси і пам'ять, які були використані під час роботи з файлом.

Приклад використання функції open():

```
file = open(r'test.txt', 'w')
print(file)
file.close()  # Закриття файлу
```

```
Результат роботи коду:
```

```
<_io.TextIOWrapper name='test.txt' mode='w' encoding='cp1251'>
```

Використання методу close() за природою пояснюється на низькому рівні, що може ускладнювати використання функції open(). Для вирішення цієї проблеми варто використовувати інший синтаксис, який включає контекстний менеджер with. Тоді файл автоматично закриється, без необхідності явно вказувати цю дію.

Приклад використання контекстного менеджеру with для роботи з файлами:

```
with open(r'test.txt', 'w') as file:
    print(file)

Результат роботи коду:
<_io.TextIOWrapper name='test.txt' mode='w' encoding='cp1251'>
```

Синтаксис із використанням with є рекомендованим, а тому надалі варто використовувати саме його.

1.4.2. Читання даних з файлу

Читання файлу за допомогою функції open() є дією за замовчуванням, проте також можна вказати цей режим явно 'r' або додати до рядка режиму символ '+' ('w+', 'a+', 'x+' тощо). При цьому слід враховувати особливості кожного режиму описані у попередньому пункті. За потреби можна перевірити можливість зчитування з файлу за допомогою методу readable(). Для зчитування даних існує три методи: read(), readline() та readlines().

За допомогою методу read() без аргументів відбувається зчитування всього вмісту файлу. Аргументом цього методу може бути будь-яке ціле число, яке вказує на кількість символів, що потрібно зчитати.

Метод readline() без аргументів зчитує один рядок. Використовуючи кілька разів цей метод відбудеться почергове зчитування рядків. Аргументом даного методу також може бути ціле число, яке відповідатиме кількості символів у цього рядку, що потрібно зчитати.

Метод readlines() без аргументів зчитує всі рядки та повертає їх у вигляді списку. Аргументом цього методу може бути будь-яке ціле число, що буде інтерпретовано за правилом: якщо число менше кількості символів у рядку, то повертається лише список із цілим рядком. Наприклад, у файлі два рядки по 7 символів, тоді від readlines(1) до readlines(7) повертатиметься список із повним першим рядком, а від readlines(8) і більше – список із двома повними рядками.

Приклад зчитування даних з файлу:

```
with open(r'test.txt') as file:
    print('Весь вміст:\n', file.read(), sep='')

print()
with open(r'test.txt') as file:
    print('Перші 4 символи:', file.read(4))
    print('Наступні 4 символи:', file.read(4))

print()
with open(r'test.txt') as file:
    print('Перший рядок:', file.readline(), end='')
    print('Другий рядок:', file.readline(), end='')
    print('Перші 7 символів третього рядка:', file.readline(7))

print()
with open(r'test.txt') as file:
```

```
    lines = file.readlines()
    print('Всі рядку у формі списку:', lines)
    print('Читання по рядках за допомогою циклу for:')
    for line in lines:
        print(line, end='')

print('\n')
with open(r'test.txt') as file:
    lines = file.readlines(17)  # у першому рядку 17 символів
    print('Використання readlines(17):', lines)

with open(r'test.txt') as file:
    lines = file.readlines(18)  # у першому рядку 17 символів
    print('Використання readlines(18):', lines)
```

```
Результат роботи коду:
Весь вміст:
Gaudeamus igitur,
Iuvenes dum sumus,
Post jucundam juventutem,
Post molestam senectutem,
Nos habebit humus.

Перші 4 символи: Gaud
Наступні 4 символи: eamu

Перший рядок: Gaudeamus igitur,
Другий рядок: Iuvenes dum sumus,
Перші 7 символів третього рядка: Post ju

Всі рядки у формі списку: ['Gaudeamus igitur,\n', 'Iuvenes dum
sumus,\n', 'Post jucundam juventutem,\n', 'Post molestam
senectutem,\n', 'Nos habebit humus.']
Читання по рядках за допомогою циклу for:
Gaudeamus igitur,
Iuvenes dum sumus,
Post jucundam juventutem,
Post molestam senectutem,
Nos habebit humus.

Використання readlines(17): ['Gaudeamus igitur,\n']
Використання readlines(18): ['Gaudeamus igitur,\n', 'Iuvenes dum
sumus,\n']
```

При повторному використанні методів читання у одному блоці with варто розуміти концепцію покажчика. Наприклад, після повторного виклику методу read() результатом буде лише порожній рядок. Це відбувається через те, що після першого виклику read() покажчик переміщується у кінець файлу.

Для визначення позиції покажчика використовується метод tell(), а для його переміщення – метод seek(n), де n – кількість символів, на яку буде зміщено покажчик.

Приклад використання переміщення покажчика:

```
with open(r'test.txt') as file:
    print('Позиція покажчика одразу після відкриття файлу:',
```

```
file.tell())
    print('Перші 4 символи:', file.read(4))
    print('Позиція покажчика після попередньої дії:', file.tell())
    print('Наступні 4 символи:', file.read(4))
    print('Позиція покажчика після попередньої дії:', file.tell())
    file.seek(4)
    print('Позиція покажчика після seek(4):', file.tell())
    print('Перші 4 символи після зміщення покажчика seek(4):',
file.read(4))
```

```
Результат роботи коду:
Позиція покажчика одразу після відкриття файлу: 0
Перші 4 символи: Gaud
Позиція покажчика після попередньої дії: 4
Наступні 4 символи: eamu
Позиція покажчика після попередньої дії: 8
Позиція покажчика після seek(4): 4
Перші 4 символи після зміщення покажчика seek(4): eamu
```

Функція open() також має інші параметри, які важливі при роботі з файлами. Наприклад, за допомогою параметра encoding можна змінити формат кодування. Наприклад, стандартне кодування функції open() ('cp1251') не може розпізнати українську мову, тому варто вказати більш універсальне кодування 'utf-8'.

Приклад використання іншого кодування:

```
with open(r'test.txt') as file:
    print('Відкрито файл із кодуванням cp1251:')
    print(file.read())

print()

with open(r'test.txt', encoding='utf-8') as file:
    print('Відкрито файл із кодуванням utf-8:')
    print(file.read())
```

```
Результат роботи коду:
Відкрито файл із кодуванням cp1251:
РћС,Р¶Рµ, РІРµСЃРµР»С–РjРsСЃСЏ,
РµРsРєРё РjРё РjРsР»РsРґС–!
РµС–СЃР»СЏ РІРµСЃРµР»РsС— РjРsР»РsРґРsСЃС,С–,
РµС–СЃР»СЏ РsР±С,СЏР¶Р»РёРІРsС— СЃС,Р°СЂРsСЃС,С–,
РќР°СЃ РїСЂРёР№РjРµ Р·РµРjР»СЏ.

Відкрито файл із кодуванням utf-8:
Отже, веселімося,
Поки ми молоді!
Після веселої молодості,
Після обтяжливої старості,
Нас прийме земля.
```

1.4.3. Запис даних у файл

Для запису у файл потрібно скористатися режимами 'w', 'x', 'a' або комбінаціями з '+', у тому числі 'r+'. При цьому слід враховувати особливості кожного режиму описані у попередньому пункті. За потреби можна перевірити можливість запису у файл за допомогою методу writable().

Процес запису можливий двома способами: через метод write() або writelines(). Метод write() приймає як аргумент лише рядки. А метод writelines() використовує для запису будь-який ітерований об'єкт, що містить рядки. Наприклад, для writelines() можна використовувати список чи кортеж із рядками.

Приклад запису у файл:

```
with open(r'test.txt', 'w') as file:
    # Запис одного рядка
    file.write("Gaudeamus")
    # Запис кількох рядків за допомогою потрійних лапок
    file.write("""
Gaudeamus igitur,
Iuvenes dum sumus,
Gaudeamus igitur,
Iuvenes dum sumus,
""")
    # Запис кількох рядків за допомогою \n
    file.write("Post jucundam juventutem,\nPost molestam
senectutem,\n")
    # Запис кількох рядків за допомогою методу writelines
    list_of_str = ["Nos habebit humus,\n", "Nos habebit humus."]
    file.writelines(list_of_str)

with open(r'test.txt', 'r') as file:
    print(file.read())
```

```
Результат роботи коду:
Gaudeamus
Gaudeamus igitur,
Iuvenes dum sumus,
Gaudeamus igitur,
Iuvenes dum sumus,
Post jucundam juventutem,
Post molestam senectutem,
Nos habebit humus,
Nos habebit humus.
```

Оскільки методи write() та writelines() використовують рядок та ітерований об'єкт із рядками, то будь-який інший тип даних потрібно попередньо перетворити за допомогою функції str(). У тому числі str() може перетворити інші типи у списку на рядки.

Приклад запису інших типів даних:

```
with open(r'test.txt', 'w') as file:
    file.write(str(3.11))
    list1 = ['Python', 3.11]
    file.writelines(str(list1))
```

```
with open(r'test.txt') as file:
    print(file.read())

Результат роботи коду:
3.11['Python', 3.11]
```

Аналогічно до читання, при записуванні даних слід враховувати кодування, оскільки українська мова не підтримується кодуванням за замовчуванням. Варто вказувати більш універсальне кодування 'utf-8'.

Приклад роботи коду з використанням різних кодувань:

```
with open(r'test.txt', 'w', encoding='utf-8') as file:
    # Запис одного рядка
    file.write("Gaudeamus")
    # Запис кількох рядків за допомогою потрійних лапок
    file.write("""
Отже, веселімося,
Поки ми молоді!
Отже, веселімося,
Поки ми молоді!
""")
    # Запис кількох рядків за допомогою \n
    file.write("Після веселої молодості,\nПісля обтяжливої
старості\n,")
    # Запис кількох рядків за допомогою методу writelines
    list_of_str = ["Нас прийме земля,\n", "Нас прийме земля."]
    file.writelines(list_of_str)

with open(r'test.txt', 'r', encoding='utf-8') as file:
    print(file.read())

Результат роботи коду:
Gaudeamus
Отже, веселімося,
Поки ми молоді!
Отже, веселімося,
Поки ми молоді!
Після веселої молодості,
Після обтяжливої старості,
Нас прийме земля,
Нас прийме земля.
```

1.4.4. Робота з файловою системою за допомогою пакету os

При роботі з даними часто виникає потреба автоматично обирати всі файли з певної папки, з конкретним розширенням тощо. Для вирішення цього питання можна скористатися можливостями пакету os (operation system).

Метод os.listdir(dir) повертає список всіх файлів та директорій у вказаній директорії dir. Якщо не вказувати аргумент dir, то повернеться список для директорії, у якій запущено скрипт.

Приклад використання os.listdir():

```
import os

print(os.listdir(r'C:\\'))
print(os.listdir(r'C:\Users\admin\Desktop\pythonProject'))
print(os.listdir())

Результат роботи коду:
['$Recycle.Bin', '$WinREAgent', 'Documents and Settings',
'DumpStack.log.tmp', 'Games', 'hiberfil.sys', 'pagefile.sys',
'PerfLogs', 'Program Files', 'Program Files (x86)', 'ProgramData',
'Recovery', 'swapfile.sys', 'System Volume Information', 'Users',
'Windows']
['.idea', 'base.py', 'create.py', 'methods.py', 'photo.py',
'test.png', 'test.py', 'venv']
['.idea', 'base.py', 'create.py', 'methods.py', 'photo.py',
'test.png', 'test.py', 'venv']
```

Для перевірки існування файлу чи директорії можна скористатись функціями os.path.isfile(path) та os.path.isdir(path). Якщо файл/директорія існує, то повернеться значення True, інакше – False.

Приклад використання os.path.isfile() та os.path.isdir():

```
import os

print('Чи існує файл base.py?', os.path.isfile('base.py'))
print('Чи існує файл base1.py?', os.path.isfile('base1.py'))
print('Чи існує директорія venv?', os.path.isdir('venv'))
print('Чи існує директорія env?', os.path.isdir('env'))

Результат роботи коду:
Чи існує файл base.py? True
Чи існує файл base1.py? False
Чи існує директорія venv? True
Чи існує директорія env? False
```

Пакет os містить багато інших функцій для роботи з файлами, зокрема:

1. os.getcwd() – повертає поточний робочий каталог.
2. os.path.join(path1, path2) – з'єднує частини шляху до файлу або каталогу, враховуючи правильний роздільник для поточної операційної системи.
3. os.mkdir(path) – створює новий каталог за вказаним шляхом.
4. os.rmdir(path) – видаляє порожній каталог за вказаним шляхом.
5. os.remove(path) – видаляє файл за вказаним шляхом.

Приклад використання функцій пакету os:

```
import os

print('Поточна директорія', os.getcwd())
print('Шлях до файлу:', os.path.join(os.getcwd(), 'base.py'))
os.mkdir('env')
print('Чи створено папку env?', os.path.isdir('env'))
os.rmdir('env')
```

```
print('Чи видалено папку env?', os.path.isdir('env'))
os.remove('test.png')
print('Чи видалено файл test.png?', os.path.isfile('test.png'))

Результат роботи коду:
Поточна директорія C:\Users\admin\Desktop\pythonProject
Шлях до файлу: C:\Users\admin\Desktop\pythonProject\base.py
Чи створено папку env? True
Чи видалено папку env? False
Чи видалено файл test.png? False
```

1.4.5. Робота з файлами, що містять табличні дані

Файли, які містять табличні дані, є важливою складовою багатьох програм та завдань у програмуванні та аналізі даних. Табличні дані зазвичай зберігаються у форматах, таких як CSV (Comma-Separated Values) і Excel, і обробляються за допомогою спеціалізованих бібліотек та модулів. У цьому розділі ми розглянемо основні концепції та операції, які дозволяють працювати з цими типами файлів у мові програмування python.

1.4.5.1. Робота з CSV-файлами

CSV-файли є одними з найпоширеніших форматів для збереження табличних даних. Вони складаються із коми та рядків, де кожен рядок представляє один рядок даних, а коми розділяють значення в рядку. Основні операції з CSV-файлами включають читання, запис і обробку даних.

Для створення CSV-файлу можна використати вбудований пакет csv:

```
import csv
import random

# Визначаємо ім'я файлу та відкриваємо його для запису
file_name = "random_data.csv"
with open(file_name, mode='w', newline='', encoding='utf-8') as
file:
    writer = csv.writer(file)

    # Записуємо заголовок CSV-файлу
    writer.writerow(["Name", "Age", "Salary"])

    # Генеруємо рандомні дані і записуємо їх у CSV
    for _ in range(10):  # Приклад для 10 рядків
        name = f"User {random.randint(1, 100)}"
        age = random.randint(18, 65)
        salary = round(random.uniform(1000, 5000), 2)
        writer.writerow([name, age, salary])

print(f"Рандомний CSV-файл '{file_name}' створено успішно.")

    Результат роботи коду:
Рандомний CSV-файл 'random_data.csv' створено успішно.
```

При цьому вміст файлу random_data.csv виглядатиме як:

```
Name,Age,Salary
User 5,42,2195.03
```

```
User 54,49,4547.56
User 73,34,4426.92
User 64,41,1517.11
User 28,23,1442.12
User 8,37,4497.13
User 72,59,2553.25
User 57,42,2837.58
User 96,58,4595.61
User 76,25,1390.45
```

Для читання даних із CSV-файлу також можна використовувати пакет csv, а для збереження даних у змінну щонайкраще використовувати numpy:

```
import csv
import numpy as np

# Читання CSV-файлу і збереження даних у списку списків
data = []
with open('random_data.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        data.append(row)

# Перетворення списку списків у масив numpy
numpy_array = np.array(data)

# Виведення масиву numpy
print(numpy_array)
```

Результат роботи коду:

```
[['Name' 'Age' 'Salary']
 ['User 5' '42' '2195.03']
 ['User 54' '49' '4547.56']
 ['User 73' '34' '4426.92']
 ['User 64' '41' '1517.11']
 ['User 28' '23' '1442.12']
 ['User 8' '37' '4497.13']
 ['User 72' '59' '2553.25']
 ['User 57' '42' '2837.58']
 ['User 96' '58' '4595.61']
 ['User 76' '25' '1390.45']]
```

При роботі з даними також зручно одразу зчитувати дані з табличного файлу у особливу структуру даних з пакету pandas – DataFrame:

```
import pandas as pd

# Читання CSV-файлу за допомогою pandas
df = pd.read_csv('random_data.csv', encoding='utf-8')

# Виведення датафрейму
print(df)
```

Результат роботи коду:

```
      Name  Age   Salary
0   User 5   42  2195.03
1  User 54   49  4547.56
2  User 73   34  4426.92
3  User 64   41  1517.11
4  User 28   23  1442.12
5   User 8   37  4497.13
6  User 72   59  2553.25
7  User 57   42  2837.58
8  User 96   58  4595.61
9  User 76   25  1390.45
```

1.4.5.2. Робота з Excel-файлами

Excel-файли часто використовуються для збереження табличних даних з багатьма аркушами. У python для роботи з Excel-файлами використовують бібліотеки, такі як openpyxl або pandas, що є надбудовою над openpyxl.

Приклад створення Excel-файлу:

```
import openpyxl
import random

# Створюємо новий Excel-файл та робочу книгу
workbook = openpyxl.Workbook()
sheet = workbook.active

# Записуємо заголовок Excel-файлу
sheet["A1"] = "Name"
sheet["B1"] = "Age"
sheet["C1"] = "Salary"

# Генеруємо та заповнюємо дані
for row in range(2, 12):  # Приклад для 10 рядків
    name = f"User {random.randint(1, 100)}"
    age = random.randint(18, 65)
    salary = round(random.uniform(1000, 5000), 2)

    sheet[f"A{row}"] = name
    sheet[f"B{row}"] = age
    sheet[f"C{row}"] = salary

# Зберігаємо Excel-файл
file_name = "random_data.xlsx"
workbook.save(file_name)

print(f"Рандомний Excel-файл '{file_name}' створено успішно.")
```

```
Результат роботи коду:
Рандомний Excel-файл 'random_data.xlsx' створено успішно.
```

При цьому вміст файлу random_data.csv виглядатиме як:

```
Name    Age Salary
```

```
User 4  31  4789,18
User 98 57  1687,78
User 84 51  3709,71
User 4  45  3533,47
User 42 22  3544,48
User 25 51  1800,22
User 70 19  2958,24
User 46 36  2079,93
User 14 38  2220,88
User 14 50  2768,32
```

Для читання даних із Excel-файлу зручно користуватись пакетом pandas, який зберігає дані у вигляді структури DataFrame:

```
import pandas as pd

# Читання CSV-файлу за допомогою pandas
df = pd.read_excel('random_data.xlsx')

# Виведення датафрейму
print(df)

# Отримання окремих колонок даних
ages = df[['Age']]
print('Вік користувачів:\n', ages)

# DataFrame підтримує дії подібні на numpy
print('Середній вік користувача:', int(ages.mean()))
```

```
Результат роботи коду:
      Name  Age   Salary
0  User 4    31  4789.18
1  User 98   57  1687.78
2  User 84   51  3709.71
3  User 4    45  3533.47
4  User 42   22  3544.48
5  User 25   51  1800.22
6  User 70   19  2958.24
7  User 46   36  2079.93
8  User 14   38  2220.88
9  User 14   50  2768.32
Вік користувачів:
    Age
0   31
1   57
2   51
3   45
4   22
5   51
6   19
7   36
8   38
9   50
```

```
Середній вік користувача: 40
```

Завдання та запитання до підрозділу «Робота з файлами»

1. Вкажіть режим відкриття файлу для запису, при якому існуючі у файлі дані не видаляються, а нові дані додаються в кінець файлу.
2. Надайте визначення режиму «w +» для відкриття файлу.
3. Надайте визначення режиму «x» для відкриття файлу.
4. Які типи даних можуть бути передані в якості аргументу функції write() при роботі з файлами?
5. Який пакет (пакети) необхідно імпортувати, щоб прочитати інформацію з файлу з розширенням .xlsx?

1.5. Функції python

У середовищі python без операцій імпорту є більше сотні вбудованих об'єктів, типів, функцій, констант [14], виключень (exception) тощо, мова в даному розділі піде про функції. Вбудовані функції python умовно можна розділити за категоріями [15, 16]:
Функції перетворення типів: int, float, bool, complex, str, list, tuple, dict, set, тощо
Функції роботи з числами і рядками: abs, round, pow, len, ord, chr, hex, divmod, unicode.
Функції обробки даних: max, min, sum, zip, range, apply, map, filter, iter, enumerate.
Функції визначення властивостей: type, hash, isinstance, issubclass.
Функції для доступу до внутрішніх структур: locals, globals, vars, intern, dir.
Функції введення-виведення: input, print, open, close.
Функції для роботи з атрибутами класів: getattr, setattr, delattr, hasattr.
Функції-декоратори методів класів: staticmethod, classmethod, property.
Можна, звернувшись за такими адресами [9, 15, 17], отримати інформацію про вбудовані функції python:

1.5.1. Визначення функції

Функції є основою при написанні програм на python. Функція – це поіменований фрагменти програми, який використовуються багаторазово і до якого можна звернутися з іншого місця програми.

Функції дозволяють дати ім'я певному блоку команд з тим, щоб надалі запускати цей блок по зазначеному імені в будь-якому місці програми і скільки завгодно разів. Це називається викликом функції. Такі функції називаються іменованими і визначаються за допомогою зарезервованого слова **def** за такою схемою:

```
def ІМ'Я_ФУНКЦІЇ (ПАРАМЕТРИ):
    """ Рядки документації """ <Тіло функції>
    return <результат>
```

Ім'я функції повинно бути унікальним ідентифікатором. Імена функціям задаються по тим же правилам, що і імена змінним (вони повинні починатися з літери або _ і містити тільки букви, цифри або _).

Функція може не містити параметри, але круглі дужки необхідно вказувати.

Таким чином при створенні функцій слід виконувати такі правила:

- Заголовок функції складається з ключового слова def, назви функції та переліку формальних параметрів, узятих у круглі дужки. Якщо функції не треба передавати значення, слід записати порожні дужки.
- Після дужок ставиться двокрапка (:), тіло функції однаковою кількістю пробілів зліва.
- Опис функції має міститися вище від виклику функції.

- Коли програма зустрічає виклик функції, керування передається у функцію із зазначеною назвою. Значення фактичних параметрів (аргументів) передаються формальним параметрам, тому слід дотримуватися відповідності між списками формальних і фактичних параметрів щодо їх кількості, порядку, типів.
- Після виконання операторів тіла функції програма продовжить виконуватися з оператора, наступного за оператором виклику функції.

1.5.2. Передача інформації функції

Функція може приймати параметри та повертати значення. Параметри функції – звичайні змінні, якими функція користується для внутрішніх розрахунків, параметри перераховуються через кому:

```
def add(x, y):
    return x + y
```

Формальні параметри – параметри, що вказуються при оголошенні функції, в даному прикладі – це x та y.

Фактичні параметри (аргументи) – параметри, що передаються в функцію при її виклику, в даному прикладі – це 10 і 21:

```
print(f'add = {add(10, 21)}')     # виклик функції add(10, 21)
Результат роботи коду:
add = 31
```

Функція може приймати будь-яку кількість аргументів (включаючи нуль) будь-якого типу. Вона може повертати будь-яку кількість значень будь-якого типу (також включаючи нуль).

При необхідності повернути результат роботи функції в програму, з якої вона викликалася, застосовується оператор return. Вираз, що стоїть після return повертається в якості результату виклику функції, як в попередньому прикладі. Всередині функції може міститися довільна кількість інструкцій return, однак спрацює лише одна з них.

Якщо функція не має аргументів, return використовується для виходу з функції, інакше вихід відбудеться при досягненні кінця функції, так як інструкція return є необов'язковою, тобто функція може і не закінчуватися інструкцією return:

```
def func1():
    print("Інструкція return в функції є необов'язковою.")

func1()           # виклик функції


Результат роботи коду:
Інструкція return в функції є необов'язковою.
```

Якщо в середині функцій немає інструкцій, то необхідно помістити оператор pass, який не виконує ніяких дій, при цьому функція поверне значення None:

```
def func2():
    # оператор pass не виконує ніяких дій, функція поверне
значення None:
    pass

   Результат роботи коду:
None
```

Крім ключового слова return, у функціях інколи застосовують yield. У python ключове слово 'yield' використовується для створення генераторів і керування потоками даних. Генератори – це спеціальний вид ітераторів [18], які дозволяють генерувати значення по одному елементу на кожному кроці [19]. Це особливо корисно в ситуаціях, коли вам потрібно обробити великий набір даних без витрати великої кількості пам'яті.

Основна ідея 'yield' полягає в тому, що він призупиняє виконання функції та повертає значення до моменту виклику наступного елементу. Це дає можливість зберігати стан функції між викликами і продовжувати виконання з місця призупинення.

Розглянемо простий приклад використання 'yield':

```
def number_generator(n):
    for i in range(n):
        yield i

# Створюємо генератор
gen = number_generator(5555)

# Витягуємо значення з генератора
sum = 0
for num in gen:
    sum += num

print(sum)
```

```
Результат роботи коду:
15426235
```

У цьому прикладі функція number_generator створює генератор, який повертає числа від 0 до n-1. При використанні цього генератора ми не створюємо великий список чисел у пам'яті, а отримуємо їх по одному, що зменшує обсяг використаної пам'яті і підвищує продуктивність програми.

Ключове слово yield також може бути використано для зчитування великих файлів по частинах, обробки потоку даних або створення нескінченних послідовностей. Він є потужним інструментом для оптимізації та управління потоками даних у програмах.

Імена функцій в python є змінними, що містять адресу об’єкта – типу функція, тому цю адресу можна привласнити іншій змінній і викликати функцію з іншим ім’ям. Функції, які передаються за посиланням називаються функціями зворотнього виклику:

```
print('1 результат функції add() ', add(10, 3))
print(f"2 результат функції add()  {add('студент ', 'КПІ')}")
ff = add   # Зберігаємо посилання на функцію add() в змінній ff
# Викликаємо функцію з іншим ім'ям
print('1.1 результат функції ff()  ', ff(10, 3))
print(f"1.2 результат функції ff()  {add('студент ', 'КПІ')}\n")
```

```
   Результат роботи коду:
1 результат функції add()  13
2 результат функції add()  студент КПІ
1.1 результат функції ff()   13
1.2 результат функції ff()  студент КПІ
```

1.5.2.1. Область видимості змінної

При представлені змінних всередині визначення функції, вони не пов’язані з іншими змінними з таким же ім’ям за межами функції – тобто імена змінних є локальними в функції. Це називається областю видимості змінної. Область видимості всіх змінних обмежена блоком, в якому вони представлені, тобто є локальні і глобальні змінні, як в наступному прикладі:

```
x = 10                # global x
def func(x):
    print('в основній програмі x = ', x)
    x = 20
    print('значення x задане в func(x) = ', x)
    global x1
    x1 = 30

func(x)
y = x+x1
print(f'значення x в основній програмі залишається незмінним x =
{x} \n'
        f'змінна x1 в func() задана як global x1, тому {x1 = }, а
y = {x + x1}')
```

```
   Результат роботи коду:
в основній програмі x =  10
значення x задане в func(x) =  20
значення x в основній програмі залишається незмінним x = 10
змінна x1 в func() задана як global x1, тому x1 = 30, а y = 40
```

1.5.2.2. Позиційні та іменовані аргументи функції

Функція може мати довільну кількість аргументів або не мати їх зовсім. В функцію можна передавати не лише окремі змінні (об’єкти) але і послідовності (рядки, списки, словники, кортежі тощо) і навіть функції. Можна використовувати функції з різними типами аргументів.

Аргументи функцій в python можуть бути:

- Позиційні (обов'язкові), які найбільш поширені, їх значення копіюються відповідно до черговості розташування параметрів;
- Іменовані (необов'язкові).

Незважаючи на поширеність позиційних аргументів, вони мають недолік, який полягає в тому, що потрібно запам’ятовувати значення кожної позиції. Щоб уникнути плутанини з позиційними аргументами, можна вказати аргументи за допомогою імен відповідних аргументів, тобто задавати іменовані аргументи. Порядок аргументів в цьому випадку можна змінювати.

Якщо викликати функцію, яка має як позиційні аргументи, так і іменовані аргументи, то позиційні аргументи необхідно вказувати першими. Якщо функція приймає більше трьох аргументів, потрібно хоча б для частини з них вказати ім'я. Особливо важливо дати назви аргументам, якщо кілька значень мають однаковий тип.

Для аргументів можна вказати значення за замовчуванням. Значення за замовчуванням використовуються в тому випадку, якщо при виклику функції не було вказано відповідний параметр. Але якщо передати інше значення параметру в функцію, він буде використаний замість аргументу за замовчуванням. Щоб зробити деякі аргументи необов'язковими їм необхідно при визначенні функції присвоїти початкові значення,

необов'язкові (іменовані) аргументи повинні слідувати за обов'язковими (позиційними аргументами).

Приклад коду використання позиційних та іменованих аргументів:

```
def func(a, b, c=2):                          # c - іменований (не обов'язковий) аргумент
    return a + b + c
print(f'func(1, 2), a + b + c = {func(1, 2)}')      # c = 2 (за замовчуванням)

print('Якщо при зверненні до функції параметру c задано нове значення, то значення аргументу c змінюється')
print(f'func(1, 2, 3), a + b + c = {func(1, 2, 3)}')

print('Коли значення задаються явно, то порядок та назви аргументів можна змінювати')
print(f'func(b=3, a=1), a + b + c = {func(b=3, a=1)}')
print(f'func(b=6, a=2, c=8), a + b + c = {func(b=6, a=2, c= 8)}')

print("Якщо  обов'язковий параметр не визначений (наприклад, b), визначати не обов'язковий параметр c не можна")
print(f'func(a=3, c=6) = {func(a=3, c=6)}\n')       # b - відсутній
```

Результат роботи коду:

```
func(1, 2), a + b + c = 5
Якщо при зверненні до функції параметру c задано нове значення, то значення аргументу c змінюється
func(1, 2, 3), a + b + c = 6
Коли значення задаються явно, то порядок та назви аргументів можна змінювати
func(b=3, a=1), a + b + c = 6
func(b=6, a=2, c=8), a + b + c = 16
```

Якщо обов'язковий параметр не визначений (наприклад, b), визначати не обов'язковий параметр c не можна

```
TypeError: func() missing 1 required positional argument: 'b'
```

Якщо для параметра визначено значення за замовчуванням, Ви можете опустити відповідний аргумент, який зазвичай включається в виклик функції. Наприклад, якщо Ви помітили, що більшість викликів функції describe_student) використовується для опису студентів КПІ, задайте змінній university_name значення за замовчуванням – 'KPI'. Тепер в будь-якому виклику функції describe_student() для студента КПІ цю інформацію можна опустити:

```
def describe_student(student_name, university_name='КПІ'):
    """Виводить інформацію про студента."""
    print('\n', student_name + university_name +  ".", '\n')

describe_student(student_name='Петро Петрович Петренко - ')
describe_student(student_name='Іван Іванович Петров - ')
describe_student(student_name='John S. Smith - ',
university_name='University of Exeter')
```

Результат роботи коду:

```
Петро Петрович Петренко – КПІ.
```

```
Іван Іванович Петров – КПІ.
John S. Smith – University of Exeter.
```

1.5.2.3. Функція з довільною кількістю неіменованих аргументів

Якщо перед аргументом при визначенні функції вказати *, то функції можна передати довільну кількість неіменованих параметрів, при цьому аргумент перед яким стоїть зірочка, наприклад, *args – це кортеж. З усіма переданими функції аргументами та зі змінною args можна працювати так як з кортежем. Це корисно при написанні функцій на зразок print(), які приймають будь-яку кількість аргументів. Якщо у функції є також обов'язкові позиційні аргументи та іменовані аргументи, то аргумент *args необхідно помістити в кінець списку.

Приклад функції з довільною кількістю неіменованих аргументів:

```
def func(a, b, *args):
    print('в функції func(a, b, *args),  a, b, *args = ', a, b,
args)
    print('функція func() повертає в програму кортеж з усіх
переданих їй аргументів \n')
    return a, b, args

tuple1 = (12, 23, 31, 'abcd')
a = 10
b = 30
func(a, b, tuple1)
```

Результат роботи коду:

```
у функції func(a, b, *args)  a, b, *args =  10 30 ((12, 23, 31,
'abcd'),)
функція func() повертає в програму кортеж з усіх переданих їй
аргументів
```

1.5.2.4. Функція з довільною кількістю іменованих аргументів

Якщо перед аргументом при визначенні функція вказати дві зірочки **, то в цьому аргументі, наприклад **keyword_args, будуть сформовані іменовані аргументи та збережені в словнику. При цьому імена аргументів стануть ключами, а їх значення – відповідними значеннями в словнику, тобто у середині функції аргумент keyword_args є словником. При цьому аргумент **keyword_args слідує за позиційними та неіменованими аргументами.

Приклад функції з довільною кількістю іменованих аргументів:

```
print('Функції з довільною кількістю іменованих аргументів')
def func(**keyword_args):
    print('Всі іменовані аргументи передані func(**keyword_args)
будуть збережені в функції як словник:')
    print('**keyword_args = ', keyword_args)
    return keyword_args
d = func(a=10, b=20, c=30)
print('і повернуті в основну програму іменовані аргументи будуть
як словник')
print(' return keyword_args ', d, '\n')
```

Результат роботи коду:

```
Всі іменовані аргументи передані func(**keyword_args) будуть
збережені в функції як словник:
```

```
**keyword_args =  {'a': 10, 'b': 20, 'c': 30}
і повернуті в основну програму іменовані аргументи будуть як
словник
 return keyword_args  {'a': 10, 'b': 20, 'c': 30}
```

Приклад функції з довільною кількістю позиційних, неіменованих та іменованих аргументів:

```
def func(n, m, dict1, *tuple1, **keyword_args):
    print('n, m, dict1, *tuple1, **keyword_args = ', n, m, dict1,
tuple1,  keyword_args )
    return n, m, dict1, tuple1, keyword_args

n = 10
m = 30
tuple1 = (1, 3, 6, 'a')
dict1 = {1: 'a', 2: 'p'}
print(f" виклик func(n, m, dict1, tuple1, a=1, b='s', c='f'):
        {func(n, m, dict1, tuple1, a=1, b='s', c='f')}\n")
```

```
     Результат роботи коду:
n, m, dict1, *tuple1, **keyword_args =
10 30 {1: 'a', 2: 'p'} ((1, 3, 6, 'a'),) {'a': 1, 'b': 's', 'c':
'f'}
виклик функції func(n, m, dict1, tuple1, a=1, b='s', c='f'):
(10, 30, {1: 'a', 2: 'p'}, ((1, 3, 6, 'a'),), {'a': 1, 'b': 's',
'c': 'f'})
```

1.5.3. Анонімні функції (lambda функція)

Анонімні функції можуть використовуватися лише один раз, при цьому виконуються вони швидше, ніж іменовані функції. Анонімні функції створюються за допомогою інструкції lambda. Крім цього, їх не обов'язково привласнювати змінній. Lambda функція може приймати будь-яку кількість аргументів, але може мати лише один вираз. При використанні функції lambda не потрібно використовувати команду повернення return, тому що lambda вже містить результат, який повертається.

Lambda функцію зручно застосовувати в таких випадках:

- якщо у невеликій ділянці коду лише один раз використовується якась функція, має сенс застосувати функцію lambda;
- якщо потрібно передати функцію як аргумент іншої функції, наприклад, функції map(), filter() або reduce(); замість визначення функції та передачі її імені, можна звернутися до lambda функції, яка визначається безпосередньо в аргументах виклику функції;
- lambda функція може викликати як вбудовані функції python (наприклад, map(), filter() або reduce()), так і іменовані функції, створені в коді.

Після того, як лямбда вираз оголошено, він може бути використаний у таких випадках:

- як функція, це випадок, коли лямбда-вираз присвоюється деякому імені, Потім за цим іменем відбувається виклик лямбда-виразу;
- як елемент (літерал) кортежу чи списку;
- як елемент (літерал) словника, який за вимогою виконує деяку дію, у цьому випадку формується так звана таблиця переходів.

Lambda-вираз відрізняється від іменованої функції за такими ознаками:

- відсутність назви;

- lambda-вираз – це функція, яка використовується тільки в одному місці;
- вони можуть містити тільки одну команду;
- їх можна передавати автоматично (змінна не потрібна);
- вони повертаються автоматично;
- синтаксис відрізняється від синтаксису іменованих функцій.

Після того, як лямбда-вираз оголошено, він може бути використаний у таких випадках:

- як функція, це випадок, коли лямбда-вираз присвоюється деякому імені, потім за цим іменем відбувається виклик лямбда-виразу;
- як елемент (літерал) кортежу чи списку;
- як елемент (літерал) словника, який за вимогою виконує деяку дію. У цьому випадку формується так звана *таблиця переходів*.

Між оголошенням та використанням лямбда-виразів (lambda) та іменованих функцій (def) існують такі відмінності:

- лямбда-вираз оголошується з допомогою ключового слова lambda, функція оголошується з допомогою ключового слова def;
- лямбда-вираз – це інструкція (іншими словами – ключове слово), def – це набір інструкцій, це означає, що лямбда-вираз може бути використаний там де не допускається використання іменованих функцій def, а саме: всередині літералів чи у викликах функцій;
- лямбда-вираз містить тільки один вираз, який повертається, результат обчислення виразу є результатом лямбда-виразу, для повернення результату з лямбда-виразу не потрібно використовувати інструкцію return;
- на відміну від оголошення функцій (def) лямбда-вирази призначені для обчислення більш простих фрагментів коду порівняно з іменованими функціями;
- програмний код лямбда-виразу формується у вигляді одного рядка, а програмний код іменованої функції може містити довільну кількість рядків.

До lambda-функції можна звертатися з іншої функції і можна звертатися через змінну, наведемо приклад:

```
# x+x - тіло функції, x - зв'язана змінна
sum_x_lambda = (lambda x: x+x)(10)
print('sum_x_lambda = ', sum_x_lambda)
print('(lambda x: x + x)(10) = ', (lambda x: x + x)(10), '\n')
      Результат роботи коду:
sum_x_lambda =  20
(lambda x: x + x)(10) =  20
```

```
      Приклад використання lambda функції з іншої функції та з
умовними операторами:

def ifelse(condition, true_part, false_part):
    if condition:
        return true_part()
    else:
        return false_part()

a = 5
b = 6
print('якщо true_part() та false_part() - функції, то \n'
      'використовуємо lambda функцію в функції ifelse()')
print('ifelse(a < b, lambda: a, lambda: b) = ', ifelse(a < b,
```

```
lambda: a, lambda: b))

print('якщо true_part та false_part – змінні, то  \n'
      'використовуємо змінні, а не lambda-функцію в функції
ifelse')
print('ifelse(a < b, a, b) = ', ifelse(a < b, a, b), '\n')
```

```
      Результат роботи коду:
Використання Lambda-функції з іншої функції та з умовними
операторами
якщо true_part() та false_part() – функції, то
використовуємо lambda функцію в функції ifelse()
ifelse(a < b, lambda: a, lambda: b) =  5
якщо true_part та false_part – змінні, то
використовуємо змінні, а не lambda-функцію в функції ifelse
ifelse(a < b, a, b) =  5
```

1.5.3.1. Використання функції lambda для роботи з ітерованими об'єктами

У вбудованій функції map(func, iterables**)** перший аргумент – це функція, другий аргумент – ітерований об'єкт, наприклад, список, функція map() повертає об'єкт типу <class 'map'>.
У вбудованій функції filter(func, iterables) також перший аргумент – це функція, другий аргумент – ітерований об'єкт, функція filter() повертає об'єкт типу <class 'filter'>.

Вбудована в python функція map (map(func, *iterables) --> map object) використовується для застосування функції (наприклад, lambda) до кожного елементу ітерованого об'єкту (наприклад, списку list1). Функція map() повертає об'єкт типу <class 'map'> (наприклад, cubed або cubed1), який ми можемо використовувати в інших частинах нашої програми. Також ми можемо передати об'єкт функції map() в функцію list() для створення ітерованого об'єкта або для друку, так як print() не працює з об'єктами типу <class 'map'> або <class 'filter'>.

Функцію lambda можна використовувати до кожного елементу ітерованого об'єкту функції map() та функції filter(), наприклад:

```
list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
# функція lambda здійснює піднесення до кубу кожного числа в
списку
cubed = map(lambda x: pow(x,3), list_1)
# функція map() повертає ітерований об'єкт cubed
print(list(cubed))
cubed1 = map(lambda x: x**3, list_1)
# функція map() повертає ітерований об'єкт cubed1
print(list(cubed1))
```

```
Результат роботи коду:
[1, 8, 27, 64, 125, 216, 343, 512, 729]
[1, 8, 27, 64, 125, 216, 343, 512, 729]
```

Наведемо також приклад використання lambda-функції для фільтрування списку:

```
list_1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
# Функція lambda фільтрує список, залишаючи в ньому тільки парні
числа
res = filter(lambda x: x % 2 == 0, list_1)
print('type(res)', type(res))
```

```
# Функції filter() повертає об'єкт класу 'filter', щоб використати функцію
# print  необхідно об'єкт <class 'filter'> перевести в список
print(list(filter(lambda x: x % 2==0, list_1)))
print(list(res))

Результат роботи коду:
type(res) <class 'filter'>
[2, 4, 6, 8]
[2, 4, 6, 8]
```

1.5.4. Створення вкладених функцій

Вкладена функція може бути будь-якої складності і повертати будь-які об'єкти (списки, кортежі, словники і навіть функції). Найпростіший приклад вкладеної функції inner_func():

```
def outer_func():
    def inner_func():
        print("Вас вітає внутрішня функція inner_func()!\n "
              "Внутрішня функція є вкладеною у зовнішню функцію
outer_func()")
    inner_func()

outer_func()

Результат роботи коду:
Вас вітає внутрішня функція inner_func()!
Внутрішня функція є вкладеною у зовнішню функцію outer_func()
```

Зовнішня функція може повертати в основну програму результат роботи внутрішньої функції таким чином:

```
def outer_func():
    def inner_func():
        print("Вас знову вітає внутрішня функція inner_func()!")
        str1 = input('Внутрішня функція просить Вас ввести будь-
який текст \n')
        return str1
    str2 = inner_func()
    return f'Зовнішня функція outer_func() повертає текст, який Ви
ввели на прохання  ' \
           f'внутрішньої функції inner_func()\n {str2}'

str3 = outer_func()
print(str3)

Результат роботи коду:
Вас знову вітає внутрішня функція inner_func()!
Внутрішня функція просить Вас ввести будь-який текст
Введений текст
Зовнішня функція outer_func() повертає текст, який Ви ввели на
прохання внутрішньої функції inner_func()
Введений текст
```

Основна особливість внутрішніх або вкладених функцій – їх здатність звертатися до змінних і об'єктів зовнішньої функції. Зовнішня функція надає простір імен, доступних для вкладеної в неї функції, наприклад:

```
def outer_func(who):
    def inner_func():
        print(f"Доброго дня, {who}")
    inner_func()

outer_func('шановні студенти.')

Результат роботи коду:
Доброго дня, шановні студенти.
```

1.5.4.1. Вкладені функції. Інкапсуляція

Найпоширеніший варіант використання внутрішніх функцій у випадку, коли потрібно захистити або приховати цю функцію від усього, що відбувається за її межами, тобто повністю приховати функцію в глобальному контексті. Таку поведінку зазвичай називають інкапсуляцією. Наприклад:

```
def increment(number):
    def inner_increment():
        return number + 1
    return inner_increment()

print(f'increment = {increment(10)}')

    Результат роботи коду:
increment = 11
```

У цьому прикладі у нас немає прямого доступу до функції inner_increment() (increment – приріст). Спробувавши звернутися до вкладеної функції, ми отримуємо NameError. Функція increment() повністю приховує функцію inner_increment(), запобігаючи доступу до неї та результатів, які вона повертає з глобальної області.

1.5.5. Збереження стану за допомогою вкладених функцій: замикання в python

Функції python в своїх правах рівні будь-яким іншим об'єктам, таким як числа, рядки, списки, кортежі, модулі і тощо. Тобто їх можна динамічно створювати або знищувати, зберігати в структурах даних, передавати в якості аргументів іншим функціям, використовувати як значення, які повертаються. В python також можна створювати функції вищого порядку, які приймають і повертають інші функції. Приклади внутрішніх функцій, які ми бачили до сих пір, були звичайними функціями, які є вкладеними всередину інших функцій. Якщо нам не потрібно приховувати ці функції, то і немає особливих причин для їх вкладення.

У наступному прикладі ми поговоримо про інший тип вкладених функцій – замикання. Замикання – це динамічно створювані функції, які повертаються іншими функціями [20]. Головна особливість полягає в тому, що функції з властивостями замикання мають повний

доступ до змінних і імен, визначених в локальному просторі імен, в якому було створено замикання, навіть якщо функція, що включає замикання, завершила своє виконання.

Щоб визначити замикання, потрібно виконати три кроки:

- створити вкладену функцію;
- у вкладеній функції здійснити операції зі змінними з зовнішньої функції;
- повернути вкладену функцію.

Замикання змушує вкладену функцію при виклику зберігати стан свого оточення. Тобто замикання – це не тільки сама внутрішня функція, а й навколишнє середовище.

Розглянемо наступний приклад:

```
# збереження стану в замиканні
def generate_power(exponent):
    def power(base):
        return base ** exponent
    return power

raise_two = generate_power(2)
raise_three = generate_power(3)
raise_two(4)
print(f'raise_two(4) = {raise_two(4)}')
raise_two(5)
print(f'raise_two(5) = {raise_two(5)}')
raise_three(4)
print(f'raise_three(4) = {raise_three(4)}')
raise_three(5)
print(f'raise_three(5) {raise_three(5)}')

Результат роботи коду:
raise_two(4) = 16
raise_two(5) = 25
raise_three(4) = 64
raise_three(5) = 125
```

В цьому прикладі ми визначаємо функцію generate_power(), яка являє собою фабрику для створення замикань та при кожному виклику створює і повертає нову функцію-замикання.

У наступному рядку цього прикладу визначається функція power(), яка є внутрішньою функцією, і приймає єдиний аргумент base і повертає результат виразу base**exponent.

Останній рядок повертає power як функціональний об'єкт, не викликаючи його. Звідки power() отримує значення показника степені exponent? Ось де в гру вступає замикання. У цьому прикладі power() отримує значення змінної exponent із зовнішньої функції generate_power().

Коли ми викликаємо generate_power():

- Визначається новий екземпляр power(), який приймає аргумент base;
- робиться «знімок» оточення power(), який включає exponent з поточним значенням;
- повертається power() разом зі станом.

Таким чином, коли ми викликаємо екземпляр power(), що повертається функцією enerate_power(), ми бачимо, що функція запам'ятовує значення степені exponent.

1.5.6. Декоратори

Найпростішою практикою налагодження коду на python є вставка функції print() в код для перевірки значень змінних. Однак, додаючи і видаляючи функції print(), ми ризикуємо забути про деякі з них. Щоб запобігти цій ситуації, ми можемо використовувати декоратори.

Декоратори python – ще один популярний і зручний варіант використання внутрішніх функцій. Декоратори – це функції вищого порядку, які приймають в якості аргументу об'єкт (функцію, метод, клас), який викликається, і повертають інший об’єкт, який теж викликається. Зазвичай декоратори застосовуються для динамічного додавання властивостей до існуючого об'єкту, який викликається, і прозорого розширення його поведінки, не зачіпаючи і не змінюючи сам об'єкт, який викликається.

Функцію-декоратор можна застосувати до будь-якого об'єкту, який викликається. Для цього в попередньому до нього рядку ставиться символ @ім’я_декоратора.
Функція any_function_used_as_decorated_function() передається як аргумент в декоратор my_decorator, якщо над нею є рядок @my_decorator.

В наступному прикладі функція-обгортка wrapper() нічого не повертає, тому в основній програмі недоступні результати роботи декорованої функції навіть, якщо б декорована функція повертала результати:

```
#конструкція декоратора
def my_decorator(decorated_function):
    def wrapper():
        print('Код до виконання декорованої функції')
        decorated_function()
        print('Код після виконання декорованої функції')
    return wrapper

#нижче наведено функцію, яка відправляється в
#декоратор в рядок decorated_function()
@my_decorator
def any_function_used_as_decorated_function():
    print('Код декорованої функції')

any_function_used_as_decorated_function()

Результат роботи коду:
Код до виконання декорованої функції
Код декорованої функції
Код після виконання декорованої функції
```

Наведемо приклад декоратора для розрахунку часу роботи будь-якої декорованої функції:

```
def time_count_decorator(my_func):
    def wrapper():
        import time
        start_time = time.time()
        result = my_func()          # виклик my_function()
        finish_time = time.time()
        t = round(finish_time - start_time, 2)
        print(f'Час виконання функції {t} секунд')
        return result
    return wrapper
```

```
@time_count_decorator
def my_function():
    a = input('Введіть число \n')
    return a

number = my_function()
print(number)

Результат роботи коду:
Введіть число
4
Час виконання функції 1.69 секунд
4
```

В цьому прикладі функція-обгортка wrapper() повертає результат роботи декорованої функції my_func, тому цей результат записується в змінну number в основній програмі при виклику декорованої функції в рядку number = my_function(). Така конструкція декоратора дозволяє повертати результати декорованої функції в основну програму.

Нижче наведено приклад декоратора для розрахунку кількості разів виклику будь-якої функції у Вашому коді:

```
def func_num_count_decorator(my_func):
    def wrapper(*args, **kwargs):
        global count
        res = my_func(*args, **kwargs)
        count = count + 1
        return res
    return wrapper

@func_num_count_decorator
def my_function1(a, b):
    return a + b

count = 0
res1 = my_function1(2, 2)
print(f'res1 при (a, b) = (2, 3): {res1}')
print(f'кількість використання функції my_function1 = {count}')
res2 = my_function1(2, 4)
print(f'res2 при (a, b) = (2, 3): {res2}')
print(f'кількість використання функції my_function1 = {count}')

    Результат роботи коду:
res1 при (a, b) = (2, 3): 4
кількість використання функції my_function1 = 1
res2 при (a, b) = (2, 3): 6
кількість використання функції my_function1 = 2
```

Наступний приклад декоратору призначено для відладки коду на python:

```
def debug(func):
    def wrapper(*args, **key_words):
```

```
        result = func(*args, **key_words)
        print(f"Функція з іменем {func.__name__} має наступні
аргументи (не словники): {args}, \n "
              f"та наступні аргументи типу словник: {key_words}.
\n"
              f" Функція з іменем {func.__name__} для зазначених
аргументів повертає результат: {result}")
        return result
    return wrapper

@debug
def add(a, b):
    return a + b

add(5, 6)
```

```
    Результат роботи коду:
Функція з іменем add має наступні аргументи (не словники): (5, 6),
та наступні аргументи типу словник: {}.
Функція з іменем add для зазначених аргументів повертає такий
результат: 11
```

Функція-декоратор debug() друкує ім'я декорованої функції (__name__ - це спеціальний метод в python), поточні значення кожного аргументу і повертає результат. Такий декоратор можна використовувати для найпростішого налагодження функцій. Як тільки ми отримуємо бажаний результат, досить видалити виклик декоратора @debug, і налагоджена функція add(a, b) буде працювати як звичайно.

1.5.7. Документування коду в python. Документування функцій

Документування коду в python – досить важливий аспект, адже від нього часом залежить читаність і швидкість розуміння Вашого коду, як іншими людьми, так і вами через деякий час.

Договір по документацію в python – PEP 257 описує угоди, пов'язані з рядками документації python, тобто описує, як потрібно документувати python код. Мета PEP 257 – стандартизувати структуру рядків документації: що вони повинні в себе включати, і як їх написати (у цілому питання синтаксису рядків документації). PEP 257 описує угоди, а не правила або синтаксис. При порушенні цих угод, найгірше, чого можна очікувати – деяких несхвальних реакцій спільноти. При цьому деякі програми, наприклад, пакет docutils, який поки що не входить в стандартну поставку python, однак про нього потрібно знати тим, хто хоче швидко написати документацію для своїх модулів. Цей пакет використовує спеціальну мову розмітки (ReStructuredText), з якої потім легко виходить документація у вигляді HTML, LaTeX та в інших форматах.

Рядок документації (docstring) – це однорядковий або багаторядковий літерал, розділений потрійними одинарними або подвійними лапками """<description>""" на початку або функції, методу, класу, модуля, пакету, який описує, що робить функція, метод, клас, модуль, пакет.

Тільки в разі, якщо рядок документації є першим оператором в пакеті, модулі, класі, методі, функції, він може бути розпізнаний компілятором байт-коду python і доступний для виконання за допомогою методу __doc__ або функції help(), наприклад:

```
import numpy
print('__doc__:', numpy.__doc__)              # документація на
пакет numpy
print('__doc__:', numpy.array.__doc__)     # документація на клас
array пакету numpy
print('__doc__:', print.__doc__)                  # документація на
функцію print пакета sys
```

Однорядкова документація повинна вміщатися на одному рядку. Закривають лапки на тому самому рядку. Порожні рядки перед або після такої документації не потрібні.

Багаторядкова документація складається: з однорядкової документації, з подальшого порожнього рядка, а потім йде більш докладний опис. Перший рядок може бути використаний автоматичними засобами індексації, тому важливо, щоб він вміщався на одному рядку і був відділений від решти документації порожнім рядком.

Перший рядок може бути на тому ж рядку, де відкриваються лапки, або на наступному рядку. Вся документація повинна мати такий же відступ, як лапки на першому рядку.

Рядки документації скрипта (самостійної програми) повинні бути доступні в якості "повідомлення щодо використання", надрукованого, коли програма викликається з некоректними або відсутніми аргументами. Такий рядок документації повинен документувати функції програми і синтаксис командного рядка, змінні оточення і файли.

Повідомлення по використанню може бути досить складним (кілька екранів) і має бути достатнім для нового користувача для використання програми належним чином, а також містити повний довідник з усіма варіантами і аргументами для досвідченого користувача.

Рядки документації функції або методу повинні узагальнити його поведінку і документувати свої аргументи, які повертаються, значення, виключення, додаткові аргументи, іменовані аргументи, і обмеження на виклик функції.

Можна додавати документацію до власних функцій, модулів, класів, заключивши рядок на початку тіла функції у лапки. Вона називається рядком документації або документаційним рядком (docstring):

```
def func(anything):
    'Функція повертає введений аргумент'
    return anything
```

Як правило документація містить розгорнуту інформацію про те, що дана функція (модуль) виконує, які аргументи приймає та що повертає в результаті виконання, опис всіх констант (функцій, для модуля). Тож документація може бути досить великого розміру і щоб використати до такої інформації форматування та вивести багато рядків коментарів необхідно заключити документаційний рядок в три пари подвійних лапок, наприклад:

```
def add(x, y):
     """Документація

       Додаткова документація
"""
    return x + y

print(add.__doc__)
print(help(add))
print(f'add = {add(1, 2)}')
```

Результат роботи коду:

```
Документація
Додаткова документація
add = 3
Help on function add in module __main__:
add(x, y)
    Документація
        Додаткова документація
None
add = 3
```

1.5.8. Функція help() в python

На відміну від звичайних коментарів, як було зазначено, до рядків документації можна звернутися під час виконання програми. Для того щоб вивести рядок документації деякої функції, необхідно викликати функцію help(), передати їй ім'я об'єкта, щоб отримати список всіх аргументів і відформатований рядок документації:
help(object) параметр: object – рядок з ім'ям об'єкта або об'єкт мови python.
опис: Функція help() викликає вбудовану довідкову систему. Ця функція призначена для інтерактивного використання.

Якщо аргумент не заданий, інтерактивна довідкова система запускається в консолі інтерпретатора.

Якщо аргумент є рядком, то відбувається пошук рядка, як ім'я модуля, класу, методу, функції, ключового слова або розділу документації, а сторінка довідки виводиться в консоль.

Якщо аргумент є будь-яким іншим типом об'єкта, генерується сторінка довідки про об'єкт.

Якщо в описі об'єкта-запиту в списку параметрів з'являється коса риска /, то це означає, що параметри до косої риски є тільки позиційними.

Нижче наведено приклад використання функції help():

```
help(print)
```

```
    Результат роботи коду:
Help on built-in function print in module builtins (довідка по
вбудованій функції print у вбудованих модулях):
print(...)
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current
sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

Функція print() має кілька "прихованих" аргументів, які задаються за замовчуванням в момент виклику:

- value – ми перераховуємо об'єкти через кому, які необхідно вивести на екран sep – вказуємо рядок-розділювач між рядками. По замовчуванню стоїть пробіл end – рядок, що розміщуємо після останнього об'єкту. По замовчуванню – перехід на новий рядок.
- File - параметр file контролює те, куди виводяться значення функції print. За замовчуванням всі значення виводяться на стандартний потік виведення – sys.stdout, тобто в console. Python дозволяє передавати file як аргумент будь-якому об'єкту з

методом write (string). За рахунок цього за допомогою print можна записувати рядки в файл.

- Flush - за замовчуванням при записі в файл або виведенні на стандартний потік виведення, інформація, яка виводиться, буферизується. Параметр flush дозволяє відключати буферизацію. Приклад скрипта, який виводить число від 0 до 10 щосекунди (файл print_nums.py).
- Буферизація (Buffer) – метод організації обміну, зокрема, введення і виведення даних в комп'ютерах та інших обчислювальних пристроях, який передбачає використання буфера для тимчасового зберігання даних. При введенні даних одні пристрої або процеси роблять запис даних в буфер, а інші – читання з нього, при виводі – навпаки. Процес, який виконав запис в буфер, може негайно продовжувати роботу, не чекаючи, поки дані будуть оброблені іншим процесом, якому вони призначені. У свою чергу, процес, який обробив деяку порцію даних, може негайно прочитати з буфера наступну порцію. Таким чином, буферизація дозволяє процесам, що забезпечують введення, виведення і обробку даних, виконуватися паралельно, не чекаючи, поки інший процес виконає свою частину роботи.

Таким чином за допомогою методу __doc__ можна отримати документацію на будь-яку функцію, модуль, клас, пакет, в тому числі написані Вами.

1.5.9. Стиль оформлення функцій

У стильовому оформленні функцій необхідно враховувати деякі правила.
Функції повинні мати змістовні імена, що складаються з літер нижнього регістру і символів підкреслення. Змістовні імена допомагають вам та іншим розробникам зрозуміти, що ж робить Ваш код. Цієї угоди слід дотримуватися і в іменах модулів.

Кожна функція повинна бути забезпечена коментарем, який коротко пояснює, що ж робить ця функція. Коментар повинен слідувати відразу ж за визначенням функції в форматі рядків документації. Якщо функція добре документована, інші розробники зможуть використовувати її, прочитавши тільки опис. Звичайно, для цього вони повинні довіряти тому, що код працює відповідно з описом, - але, якщо знати ім'я функції, які аргументи їй потрібні і яке значення вона повертає, вони зможуть використовувати її в своїх програмах.

Якщо для параметра задається значення за замовчуванням, ліворуч і праворуч від знаку рівності не повинно бути пробілів: def ім’я_функції(параметр_0, параметр_1='значення_за_замовчуванням’).

Ті ж угоди повинні застосовуватися для іменованих аргументів на виклики функцій:
ім’я_функції(значення_0, параметр_1='значення').

Документ PEP 8 (https://www.python.org/dev/peps/pep-0008/) рекомендує обмежити довжину рядків коду 79 символами, щоб рядки були повністю видно в вікні редактора нормального розміру. Якщо через параметри довжина визначення функції перевищує 79 символів, натисніть Enter після відкритої круглої дужки в рядку визначення функції. У наступному рядку двічі натисніть Tab, щоб відокремити список аргументів від тіла функції, яке повинно бути забезпечено відступом тільки на один рівень. Багато редакторів автоматично вирівнюють додаткові рядки параметрів по відступам, встановленим в першому рядку:
def ім’я_функції(параметр_0, параметр_1, параметр_2, параметр_3, параметр_4): тіло функції.

Якщо програма або модуль складається з декількох функцій, ці функції можна розділити двома порожніми рядками. Так вам буде простіше побачити, де закінчується одна функція і починається інша.

Всі команди import слід записувати на початку файлу. У цього правила є тільки один виняток: файл може починатися з коментарів, що описують програму в цілому.

Завдання та запитання до підрозділу «Функції python»

1. Напишіть код для створення функції addition(), яка приймає змінну кількість аргументів та виводить суму їх значень.
2. Чи має назву lambda-функція?
3. Чи обов'язково передавати функції значення іменованих аргументів?
4. Чи обов'язково передавати функції значення позиційних аргументів?
5. Яка конструкція декоратора?
6. Чи може функція передаватися як аргумент функції?
7. Як роздрукувати документацію функції?

1.6. Класи в python

1.6.1. Поняття про класи в python

Класи об'єднують дані і функції в один зручний об'єкт, з яким Ви можете працювати ефективно [21]. В об'єктно-орієнтованому програмуванні Ви пишете класи, що описують реально існуючі предмети і ситуації, а потім створюєте об'єкти на основі цих описів. Класи можна охарактеризувати таким чином:

- Клас – це шаблон об'єкту.
- При написанні класу визначається загальна поведінка для всіх екземплярів класу.
- Кожен екземпляр класу автоматично наділяється загальною поведінкою.
- Після цього Ви можете наділити кожен екземпляр класу унікальними особливостями на свій вибір.

Створення об'єкта на основі класу називається створенням екземпляру класу; таким чином, Ви працюєте з екземплярами класу.

- У даному розділі підручника буде розглянуто
- як писати класи і створювати екземпляри цих класів.
- приклади інформації, яка може зберігатися в екземплярах (примірниках) класу, та дій, які можуть виконуватися з екземплярами.
- як писати класи, що розширюють функціональність існуючих класів; це дозволяє організувати ефективне спільне використання коду схожими класами.
- як зберігати класи в модулях та імпортувати класи, які написані іншими програмістами, у Ваші програмні файли.

1.6.2. Створення і використання класу

Класи дозволяють моделювати практично все. Почнемо з написання класу Student, що представляє студента – не якогось конкретного, а студента взагалі. Що ми знаємо про студентів? У них є прізвище, ім'я і оцінки (бали). Також відомо, що більшість студентів відвідують лекції і здають екзамени. Ці два види інформації (прізвище, ім'я та бали) і два види поведінки (відвідувати лекції і здавати екзамени) будуть включені в клас Student, тому що вони є загальними для більшості студентів.

Клас повідомляє python, як створити об'єкт, який представляє студента. Після того як клас буде написаний, ми використовуємо його для створення екземплярів, кожен з яких представляє одного конкретного студента.

Все в python є об'єктами. Як вже зазначалося, клас – це проєкт або іншими словами шаблон об'єкта. Щоб створити клас необхідно:

- Використати ключове слово class, за яким слідує найменування класу, яке має починатися з великої літери.
- Відкрити круглі дужки, за якими слідує слово object, і закрити дужки. «Object» – це те, на чому заснований клас, або успадковується від нього. «Object» називається базовим класом або батьківським класом.

- Використати особливий метод, який є у класів, під назвою __init__. Цей метод викликається кожен раз, коли Ви створюєте об'єкт (створюєте екземпляр) на основі цього класу.
- Метод __init__ записується в класі тільки один раз, і не може бути викликаний знову всередині програми. Інше визначення методу __init__ – це конструктор, проте цей термін рідко використовується в python.

  Наприклад, нижченаведений код створює клас Student:

```
class Student(object):
    """docstring"""
    def __init__(self):
        """Constructor"""
        pass
```

В python 3 не обов'язково прямо вказувати об'єкт, тобто в python 3 круглі дужки не потрібні, коли ми формуємо наш клас.

За загальноприйнятими угодами імена класів в python починаються з символу верхнього регістру без пробілів. Після визначення класу необхідно включити рядок документації """docstring""" з коротким описом класу.

Ви можете запитати, чому функція всередині класу називається методом, а не функцією?

Функція змінює свою назву на метод, коли вона знаходиться всередині класу. Зверніть увагу на те, що кожен метод класу повинен мати як мінімум один аргумент, що не є обов'язковим для функції.

Нижченаведений код створює клас Student_of_KPI, але при цьому немає дужок зі словом object:

```
class Student_of_KPI:
    """docstring"""
    def __init__(self):
        """Constructor"""
        pass
```

1.6.3 Метод __init __() та параметр self

В даному прикладі ми додали до класу три атрибута і два методи:

```
class Student_of_KPI:
    """docstring -  model of a student of KPI"""
    def __init__(self, family_name, name, scores):
        """ docstring """
        self.family_name = family_name
        self.name = name
        self.scores = scores
    def pass_exam(self):
        """         Obtain scores at exam          """
        return "Passing exam"
    def attend_lecture(self):
        """        Attend lecture
        """
        return "I'm attending lecture!"
```

Ці три атрибути є:

self.family_name= family_name
self.name = name
self.scores= scores

Атрибути описують студента. У нього є прізвище, ім'я та бали. Також у нього є два методи. Метод описує, що робить клас. У нашому випадку, студент може здавати екзамен і відвідувати лекції.

Ви могли помітити, що всі атрибути і методи класу, мають обов'язковий аргумент self. Все, що Ви дізналися раніше про функції, також відноситься і до методів; єдина практична відмінність – спосіб виклику методів.

Метод __init __ () – це спеціальний метод, який автоматично виконується при створенні кожного нового екземпляра на базі якогось класу. Ім'я методу починається і закінчується двома символами підкреслення; ця схема запобігає конфліктам імен стандартних методів python і методів Ваших класів.

В прикладі класу Student_of_KPI метод __init__() визначається з чотирма параметрами: self, name, family_name і scores. Параметр self обов'язковий у визначенні методу класу. Він повинен передувати всім іншим параметрам. Він повинен бути включений при визначенні методів класу. При кожному виклику методу, пов'язаному з класом, автоматично передається self – посилання на екземпляр класу, він надає конкретному екземпляру класу доступ до атрибутів і методів класу. Будь-яка змінна з префіксом self доступна для всіх методів класу, і Ви також зможете звертатися до цих змінних в кожному екземплярі, створеному на основі класу. Конструкція self. family_name = family_name бере значення, що зберігається в параметрі family_name, і зберігає його у змінній (атрибуті) family_name, яка потім зв'язується зі створюваним екземпляром.

У класі Student_of_KPI також визначаються два методи: attend_lecture() і pass_exam(). Цим методам не потрібна додаткова інформація (ім’я, прізвище, бали), вони в класі визначаються параметром self, тому екземпляри класу Student_of_KPI, які будуть створені пізніше, зможуть викликати ці методи.

1.6.4. Створення екземплярів класу

Коли Ви створюєте екземпляр класу Student_of_KPI, python викликає метод __init__() з класу Student_of_KPI. Ми передаємо Student_of_KPI() прізвище, ім'я та бали в аргументах; значення self передається автоматично, так що його передавати не потрібно. В наступному прикладі ми створили три екземпляри класу Student_of_KPI:

```
class Student_of_KPI:
    """docstring"""
    def __init__(self, name, family_name, scores):
        """Constructor"""
        self.family_name = family_name
        self.name = name
        self.scores = scores
    def pass_exam(self):
        """Obtain scores at exam"""
        print("Passing exam")
        return
    def attend_lecture(self):
        """Attend lecture"""
        print("I'm attending lecture!")
        return
student1 = Student_of_KPI('Petro', 'Petrenko', [65, 94, 80])
print(f'student1.scores = {student1.scores}')
student2 = Student_of_KPI('Igor', 'Ivanchenko', [65, 94, 80])
```

```
print(f'student2.family_name = {student2.family_name}')
student3 = Student_of_KPI('Петро', 'Петренко', 75)
print(f'student3.scores = {student3.scores}')
student3.scores = [65, 94, 80]
print(f'student3.scores = {student3.scores}')
print(f'student3.family_name = {student3.family_name}')
student3.family_name = 'Іванченко'
print(f'student3.family_name = {student3.family_name}')
student1.pass_exam()
student3.attend_lecture()
```

Кожен раз, коли Ви захочете створити екземпляр на основі саме класу Student_of_KPI, необхідно передати значення аргументів family_name, name та scores:

```
student1 = Student_of_KPI('Petro', 'Petrenko', 80)
student2 = Student_of_KPI('Igor', 'Ivanov', [65, 94, 80])
student3 = Student_of_KPI('Петро', 'Петренко', 75)
```

1.6.5. Атрибути класу

Для звернення до атрибутів екземпляру класу використовується «точковий» запис. Наприклад, ми звертаємося до значення атрибута family_name екземпляру student2:
student2.family_name

Точковий запис часто використовується в python. Цей синтаксис показує, як python шукає значення атрибутів. В даному випадку python звертається до екземпляру student2 і шукає атрибут family_name, пов'язаний з екземпляром student2. Це той же атрибут, який позначався self.family_name в класі Student_of_KPI. Той же прийом використовується для роботи з атрибутом score:
student1.score()
записує бали першого студента [65, 94, 80] (значення атрибута score екземпляра student1).

Після створення екземпляру класу на основі класу Student_of_KPI можна застосовувати точковий запис для виклику будь-яких методів, визначених у Student_of_KPI:
Щоб викликати метод, вкажіть екземпляр (в даному випадку student1, student2 або student3), і метод, розділивши їх крапкою. В ході виклику методу student1.pass_exam() python шукає метод pass_exam() в класі Student_of_KPI і виконує його код. Рядок student1.attend_lecture() інтерпретується аналогічним чином. Тепер екземпляр student1 виконує отримані команди: «Passing exam» та «Attending lecture». Це дуже корисний синтаксис.

Якщо атрибутам і методам були присвоєні змістовні імена (наприклад, name, family_name, score, pass_exam() і attend_lecture()), розробник зможе легко зрозуміти, що робить блок коду, навіть якщо він бачить цей блок вперше.

У наведеній реалізації метод area отримує доступ до атрибутів width і height для розрахунку площі self.width * self.height:

```
class Rectangle:
    def __init__(self, width, height):
        print("Hello from __init__")
        self.width = width
        self.height = height
    def area(self):
        return self.width * self.height
rect = Rectangle(10, 20)
print(f'rect.width = {rect.width}')
print(f'rect.height = {rect.height}')
print(f'rect area = {rect.area()}')
```

Якби в якості параметру методу area не було вказано self, то при спробі викликати area програма була б зупинена з помилкою.

```
   Результат роботи коду:
Hello from __init__
rect.width = 10
rect.height = 20
rect.area = 200
```

Клас, як вже зазначалося, містить атрибути і методи. Атрибут може бути статичним і динамічним. Суть в тому, що для роботи зі статичним атрибутом, вам не потрібно створювати екземпляр класу, а для роботи з динамічним – потрібно.

Розглянемо приклад коду:

```
class Rectangle:
    default_color = "green"
    def __init__(self, width, height):
        self.width = width
        self.height = height
print(f'Rectangle.default_color = {Rectangle.default_color}')
rect = Rectangle(10, 20)
print(f'rect.width = {rect.width}')
```

```
   Результат роботи коду:
rectangle.default_color = green
rect.width = 10
```

В попередньому прикладі статичний атрибут: default_color. Доступ до нього можна отримати, не створюючи об'єкт класу Rectangle. Динамічні атрибути: width і height, при їх створенні було використано ключове слово self. Для доступу до width і height попередньо потрібно створити об'єкт класу Rectangle. Якщо до динамічного атрибуту класу звернутися через назву класу, то отримаємо помилку. Звертатися до статичного атрибуту default_color можна як через ім'я класу Rectangle, так і через ім'я екземпляру класу, але рівень звернення до ім'я класу Rectangle вище, тобто екземпляр класу підпорядковується класу, як в наступному прикладі:

```
# змінюємо статичний атрибут через і’мя класу
print(f'3 Rectangle.default_color = {Rectangle.default_color}')
# через ім'я класу Rectangle значення статичного атрибуту
зміниться
Rectangle.default_color = 'white'
print(f'4 Rectangle.default_color = {Rectangle.default_color} \n')
# змінюємо статичний атрибут через ім'я екземпляру класу
print(f'5 rect.default_color = {rect.default_color}')
rect.default_color = 'blue'
print(f'6 rect.default_color = {rect.default_color}')
print(f'7 Rectangle.default_color = {Rectangle.default_color}')
```

```
   Результат роботи коду:
3 Rectangle.default_color = green
4 Rectangle.default_color = white
5 rect.default_color = white
6 rect.default_color = blue
7 Rectangle.default_color = white
```

1.6.6. Методи класів

Розглянемо класифікацію методів класів:

- звичайні методи (методи екземпляру) (instance methods) без використання декораторів;
- методи рівня класу (class methods), з використання вбудованого декоратора @classmethod;
- статичні методи (static methods) з використання вбудованого декоратора @staticmethod.

В наступних прикладах ми будемо працювати не з екземплярами класів, а з методами класів, об'являємо метод класу: ex_method(self). Для виклику звичайного методу потрібен об’єкт – екземпляр класу (inst):

```
class MyClass:
    def ex_method(self):
        print("method")

inst = MyClass()
inst.ex_method()
# Звичайний метод не можна викликати через імя класу,
# Python поверне помилку: TypeError: ex_method() # missing 1
required positional argument: 'self'
# MyClass.ex_method()
   Результат роботи коду:
method
```

Для виклику звичайного методу потрібен об’єкт – екземпляр класу. Це основний найбільш поширений для використання метод. Звичайний метод приймає один обов’язковий параметр self, який вказує на екземпляр класу MyClass під час виклику методу (але, звичайно, методи екземплярів можуть приймати більше одного параметра). За допомогою параметра self методи екземплярів можуть вільно отримувати доступ до атрибутів та методів того самого об’єкта класу. Це дає їм велику силу, коли справа доходить до зміни стану об’єкта.

Методи екземплярів не лише можуть змінювати стан об’єкта, а також можуть отримати доступ до самого класу через атрибут self.__class__, тобто методи екземплярів також можуть змінювати стан класу.

Для створення методу рівня класу використовується декоратор @classmethod – це вбудований у python декоратор функції. Метод рівня класу отримує клас як неявний перший аргумент cls, так само як метод екземпляра отримує екземпляр self в якості обов'язкового першого аргументу.

Метод класу – це метод, який прив’язаний до класу, а не до об’єктів (екземплярів) класу. Метод класу має доступ до стану класу, оскільки він приймає параметр класу, який вказує на клас, а не на екземпляр класу. Метод класу може змінювати стан класу, який застосовуватиметься до всіх екземплярів класу. Наприклад, він може змінити змінну класу, яка буде застосовна до всіх екземплярів. Зазвичай ми використовуємо метод рівня класу для створення методів, які повертають об’єкти класу (подібно до конструктора) для різних випадків використання.

Для створення статичного методу використовується декоратор @staticmethod – це вбудований у python декоратор функції. Статичний метод не потребує обов'язкових параметрів. Загалом, статичні методи нічого не знають про стан класу. Це методи приймають деякі необов'язкові параметри та працюють з ними. Статичний метод також є методом, який прив’язаний до класу, а не до об’єкта класу. Статичний метод не може отримати доступ до класу або змінити стан класу. Зазвичай ми використовуємо статичні методи для створення методів типу звичайних функцій.

Наведемо приклад використання методів класу різного типу.

```
import datetime

class Student:
    def __init__(self, first_name, last_name, year_of_birth):
        self.first_name = first_name
        self.last_name = last_name
        self.nickname = None
        self.year_of_birth = year_of_birth
    def set_nickname(self, name):       # встановити псевдонім
        self.nickname = name\
    @classmethod
    def get_from_string(cls, name_string: str):  # отримати з
рядка
        first_name, last_name = name_string.split()
        return Student(first_name, last_name, 2002)
    @staticmethod
    def get_age(year_of_birth):             # встановити вік
        return datetime.datetime.now().year - int(year_of_birth)
```

В цьому прикладі:

- Звичайний метод (метод екземпляру) set_nickname() для призначення псевдоніму студенту.
- Метод рівня класу get_from_string() для створення екземпляру класу, якщо ім’я та прізвище студента введені в одному рядку через пробіл та з роком народження = 2002 за замовчуванням. Цей метод класу в певному сенсі є альтернативою конструктору об'єкта класу __init__.
- Статичний метод get_age() для розрахунку віку студента за заданим роком народження.

Нижче наведено приклад коду з використанням методів попередньо створеного класу Student:

```
s = Student.get_from_string("Сергій Шевченко")
print(s.first_name)
print(s.last_name)
print(s.year_of_birth)
s.set_nickname('Bond 007')
print(s.nickname)
print(f"Вік студента {s.first_name} {s.last_name}, "
      f"з псевдонімом {s.nickname} "
      f"{s.get_age(s.year_of_birth)} рок(и)ів.")
# не можна викликати звичайний метод через ім'я класу
s2 = Student.set_nickname('yang')
#TypeError: set_nickname() missing 1
#required positional argument: 'name'
```

Результат роботи коду:

```
Сергій
Шевченко
2000
Bond 007
Вік студента Сергій Шевченко, з псевдонімом Bond 007 22 рок(и)ів.
```

1.6.7. Порівняння методу __new__ та методу __init__

Python має особливий тип методів, які називаються магічними методами, іменами яких є подвійне підкреслення перед і в кінці. Коли ми говоримо про метод __new__, нам також потрібно говорити про метод __init__. Ці методи використовуються під час створення екземпляра класу.

За допомогою методу __new__ Ви можете налаштувати створення екземпляру класу. Цей метод використовується першим, а потім метод __init__ буде викликано для ініціалізації екземплярів класу. Метод __new__ прийматиме посилання на клас як перший аргумент, за яким йдуть аргументи, які передаються конструктору __init__. Метод __new__ відповідає за створення екземпляра, тому Ви можете використовувати цей метод для налаштування створення об'єкта. Метод __init__ буде викликано, коли __new__ метод завершить виконання.

Змінні екземпляра є локальними для екземпляра. Тому все, що Ви робите в __init__, є локальним лише для цього екземпляра.

Все, що Ви робите в __new__ , вплине на всі екземпляри, створені для цього класу.

Подібність методів__new__ та __init__(Таблиця 1.6):

- Обидва викликаються під час створення екземпляру класу.
- Відмінність методів __new__ та __init__ (Таблиця 1.6):
- Метод __new__ викликається першим при створенні екземпляра. Це відбувається перед ініціалізацією класу методом __init__.

Першим аргументом __init__ завжди є self, який є екземпляром класу. Self – це те, що повертається методом __new__. Методом __new__ передбачається повернення екземпляра класу. Якщо __new__ нічого не повертає, то __init__ не викликається.

Таблиця 1.6. Порівняння методів__new__ та __init__

	__new__	**__init__**
1	Викликається перед **__init__**	Викликається після **__new__**
2	Приймає клас як перший аргумент **cls**	Приймає екземпляр як перший аргумент **self**
3	Має повернути екземпляр отриманого класу	Нічого не повертає
4	Використовується для керування створенням екземпляра	Використовується для ініціалізації атрибутів екземпляра

Одним із найбільш наочних випадків використання методу __new__ є створення класу Singleton. Клас Singleton гарантує, що клас має лише один екземпляр, і забезпечує глобальний доступ до нього. Наприклад Singleton використовується у програмуванні ігор, де є лише один екземпляр гравця. Неважливо, скільки екземплярів Ви створите, у вас залишиться лише один.

Давайте подивимося, як досягти подібної поведінки в python:

```
class Singleton:
    instance = None     # екземпляр класу
    def __new__(cls):
        if cls.instance is None:
            print("створюємо екземпляр класу")
            # Створення нового екземпляру класу
```

```
            cls.instance = object.__new__(cls)
        return cls.instance

s1 = Singleton()
s2 = Singleton()
print(s1)
print(s2)
```

```
   Результат роботи коду:
створюємо екземпляр класу
<__main__.Singleton object at 0x000001EA2ADE5250>
<__main__.Singleton object at 0x000001EA2ADE5250>
```

В попередньому прикладі коду object – базовий клас ієрархії класів. Метод .__new__(cls) класу object під час виклику не приймає аргументів, а повертає новий безфункціональний (без атрибутів і методів) екземпляр класу.

Наступний приклад коду реалізує задачу створення класу LimitedInstances, який приймає 1 обов'язковий параметр – максимальну кількість екземплярів класу, які можливо створити. З основної програми передається максимальна кількість екземплярів класу = 5. Якщо спробувати створити 7 екземплярів класу, то легко переконатися, що створюється тільки 5:

```
class LimitedInstances:
    __instance = None
    number_of_instances = 0

    def __new__(cls, max_number_of_instances):
        cls.max_number_of_instances  = max_number_of_instances
        if cls.number_of_instances < cls.max_number_of_instances:
            print("creating...")
            cls.__instance = object.__new__(cls)
            cls.number_of_instances = cls.number_of_instances + 1
        return cls.__instance

s1 = LimitedInstances(5)
print(LimitedInstances.number_of_instances)
s2 = LimitedInstances(5)
print(LimitedInstances.number_of_instances)
s3 = LimitedInstances(5)
print(LimitedInstances.number_of_instances)
s4 = LimitedInstances(5)
print(LimitedInstances.number_of_instances)
s5 = LimitedInstances(5)
print(LimitedInstances.number_of_instances)
s6 = LimitedInstances(5)
print(LimitedInstances.number_of_instances)
s7 = LimitedInstances(5)
print(LimitedInstances.number_of_instances)

print(s1)
print(s2)
print(s3)
```

```
print(s4)
print(s5)
print(s6)
print(s7)
```

1.6.8. Спадкування

Робота над новим класом в багатьох випадках не починається з нуля. Якщо клас, який Ви пишете, є спеціалізованою версію раніше написаного класу, Ви можете скористатися спадкуванням. Клас, що успадковується від іншого, автоматично отримує всі атрибути і методи першого класу. Вихідний клас називається батьківським класом, а новий клас – класом-нащадком. Клас-нащадок успадковує атрибути і методи батьківського класу, але при цьому також може визначати власні атрибути і методи.

Можна здійснити успадкування батьківської функції __init__() класом нащадком двома способами:

- 1-ий спосіб. Щоб зберегти успадкування батьківської функції __init__(), можна додати виклик до батьківської функції __init__() в дочірньому класі.
- 2-ий спосіб. У python є функція super(), яка змусить дочірній клас успадкувати всі методи та властивості свого батьківського класу.

Наприклад:

```
# 1-ий спосіб.
class Student(Person):
    def __init__(self, name, age):
        Person.__init__(self, name, age)
st1 = Student("Шевченко", 21)

print(st1)
print(st1.name, '', st1.age)
```

```
   Результат роботи коду:
Призвище Шевченко, вік 21
Шевченко  21
```

```
# 2-ий спосіб.
class Student(Person):
    def __init__(self, name, age):
        super().__init__(name, age)

st2 = Student("Петренко", 28)
print(st2)
print(st2.name, '', st2.age)
```

```
   Результат роботи коду:
Призвище Петренко, вік 28
Петренко  28
```

1.6.9. Поліморфізм

Поліморфізм – принцип ООП, який дозволяє за одним іменем метода закріпити декілька різних алгоритмів та реалізацій. Наприклад, метод, який називається area, може мати декілька реалізацій, наприклад, здійснювати розрахунок площі предмету у формі прямокутника, квадрата, кола тощо (Рисунку 1.13).

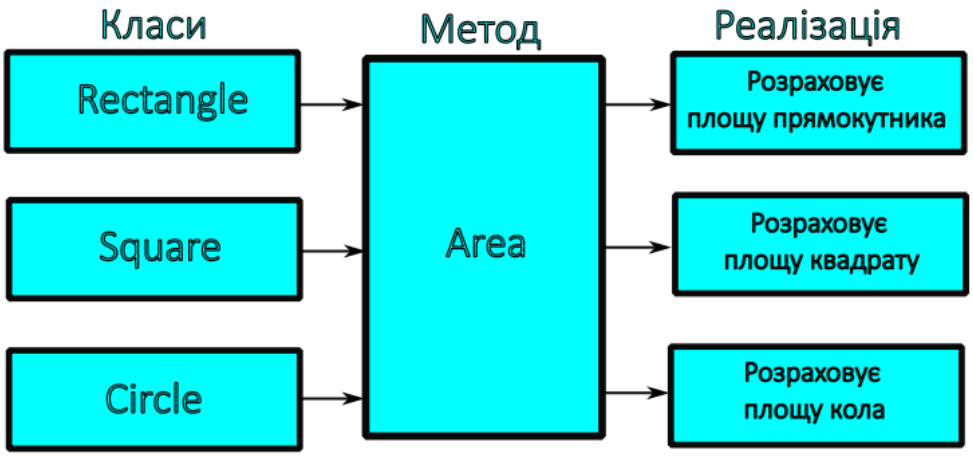


Рисунок 1.13 – Схематична ілюстрація принципу поліморфізму для розрахунку площі предмету у формі прямокутника, квадрата, кола.

Схема принципу поліморфізму реалізована для розрахунку площі предмету у формі прямокутника, квадрата, кола в наступному коді:

```
class Rectangle:
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def area(self):
        return self.a * self.b

class Square:
    def __init__(self, c):
        self.c = c

    def area(self):
        return self.c**2

class Circle:
    def __init__(self, r):
        self.r = r
    def area(self):
        return self.r **2 * 3.14

rec = Rectangle(2,3)
squ = Square(2)
cir = Circle(2)
# поліморфізм
list_ob = [rec, squ, cir]
for ob in list_ob:
    print(ob.area())

   Результат роботи коду:
6
4
12.56
```

1.6.10. Поняття про модулі. Зберігання класів та функцій в модулях

Модулі та пакети значно спрощують роботу програміста. Класи і функції, якими доводиться часто користуватися можна упакувати в модуль, і, в подальшому, завантажувати його в свої програми при необхідності [22].

Під модулем в python розуміється файл з розширенням ру. Модулі призначені для того, щоб в них зберігати функції, класи, константи тощо, які часто використовуються.

Можна умовно розділити модулі та програми: програми призначені для безпосереднього запуску, а модулі для імпортування їх в інші програми.

Варто зауважити, що модулі (в проєкті) можуть бути написані не тільки на мові python, але і на інших мовах (наприклад С).

Одна з переваг класів і функцій полягає в тому, що вони відокремлюють блоки коду від основної програми. Якщо для класів і функцій були обрані змістовні імена, Ваша програма буде набагато простіше читатися. Можна піти ще далі і зберегти класи і функції в окремому файлі, який називається модулем, а потім імпортувати модуль в свою програму. Команда import повідомляє python, що код модуля повинен бути доступний в поточному програмному файлі, який виконується. Зберігання функцій в окремих файлах дозволяє приховати другорядні деталі коду і зосередитися на логіці більш високого рівня.

Крім того, класи і функції можна використовувати в безлічі різних програм. А вміння імпортувати класи і функції дозволить вам використовувати бібліотеки функцій, написані іншими програмістами. Існує кілька способів імпортування модулів; всі вони коротко розглядаються нижче.

Модуль являє собою файл з розширенням ру, що містить код, який Ви хочете імпортувати в свою програму.

Давайте створимо модуль work_with_modul.py з функцією describe_student(). Тепер створюємо окремий файл з ім’ям import_module.py в одному каталозі з файлом work_with_module.py. Програма імпортує тільки що створений модуль work_with_module, а потім створює екземпляр класу та викликає метод класу describe_student():

```
import work_with_module       # імпортуємо модуль
student1 = work_with_module.Student('Шевченко', 20, 'КПІ')
student1.describe_student()

   Результат роботи коду:
Студент Шевченко навчається в КПІ.
Вік студента 20.
```

Також можливо імпортувати конкретний клас з модуля. Загальний синтаксис виглядає так:

from ім’я_модуля import імя_класу

Ви можете імпортувати будь-яку кількість класів або функцій з модуля, розділивши їх імена комами:

from ім’я_модуля import функція_0, функція_1, функція_2 (абоклас_0, клас_1, клас_2)

Якщо обмежитися імпортуванням тільки того класу, який Ви маєте намір використовувати, приклад work_with_ modul.py буде виглядати так:

```
from work_with_ module import Student       # з модуля імпортуємо
клас
student2 = Student('Шевченко', 20, 'КПІ')
student2.describe_student()
```

При такому синтаксисі використовувати точковий запис при створенні екземпляру класу не обов'язково. Так як клас явно імпортується в команді import, при використанні його можна викликати прямо по імені.

Також можна вказати python імпортувати всі функції та класи з модуля. Для цього використовується оператор *:

```
from work_with_module import *
```

Зірочка в команді import наказує python скопіювати кожен клас або функцію з модуля work_with_ module в програму. Після імпортування всіх функцій Ви зможете викликати кожну функцію по імені без точки перед її іменем в зверненні до неї. Проте краще не використовувати цей спосіб з великими модулями, написаними іншими розробниками. Якщо модуль містить функцію, ім'я якої збігається з існуючим ім'ям з Вашого проєкту, можливі несподівані результати. Python виявляє кілька функцій або змінних з однаковими іменами, і замість імпортування всіх функцій окремо відбувається заміна цих функцій. У таких ситуаціях найкраще імпортувати тільки потрібну функцію або функції або ж імпортувати весь модуль з подальшим застосуванням звернення до функції через точку перед її іменем. При цьому створюється код, який легко Ім’я, обране користувачем, також можна призначити для всього модуля. Призначення короткого імені для модуля - скажімо, wwp для work_with_ module дозволить вам швидше викликати функції модуля.

Виклик wwp.Student('Ivanenko', 25, 'КПІ') виходить більш компактним, ніж work_with_modul.Student('Ivanenko', 25, 'КПІ') читається і добре зрозумілий:

```
import work_with_module as wwp                    # імпортуємо модуль під псевдонімом
student3 = wwp.Student('Ivanenko', 25, 'КПІ')
student3.describe_student()
```

Модулю work_with_ module в команді import призначається псевдонім wwp, але всі класи та функції модуля зберігають свої вихідні імена.
Загальний синтаксис виглядає так:
import ім’я_модуля as псевдонім

1.6.11. Стиль оформлення класів

У стильному оформленні класів є кілька моментів, про які варто згадати окремо, особливо з ускладненням Ваших програм:

- Імена класів повинні записуватися за такою схемою: перша буква кожного слова записується в верхньому регістрі, слова не розділяються пробілами.
- Імена екземплярів класів і модулів записуються в нижньому регістрі з поділом слів символами підкреслення.
- Кожен клас повинен мати рядок документації, який слідує відразу ж за визначенням класу. Рядок документації повинен містити короткий опис того, що робить клас, і в ньому повинні дотримуватися ті ж угоди щодо форматування, які Ви використовували при написанні рядків документації у функціях.
- Кожен модуль також повинен містити рядок документації з описом можливих застосувань класів в модулі.
- Порожні рядки можуть використовуватися для структурування коду, але зловживати ними не варто. У класах можна розділяти методи одним порожнім рядком, а в модулях для поділу класів – два порожні рядки.

Якщо вам буде потрібно імпортувати модуль зі стандартної бібліотеки і модуль з бібліотеки, написаної вами, почніть з команди import для модуля стандартної бібліотеки. Потім додайте порожній рядок і команду import для модуля, який написаного вами. У програмах з декількома командами import виконання цієї угоди допоможе зрозуміти, звідки беруться різні модулі, що використовуються в програмі.

Завдання та запитання до підрозділу «Класи в python»

1. Дайте визначення класу.
2. Як роздрукувати документацію класу?
3. Напишіть код, в якому створіть клас під назвою Student із двома статичними атрибутами student_id, student_name. Створіть функцію для друку всіх атрибутів класу Student.
4. Напишіть код, в якому створіть клас під назвою Student із двома статичними атрибутами student_id, student_name. Створіть функцію для друку всіх атрибутів класу Student.
5. Напишіть код, в якому створіть клас N_samples. Зробіть так, щоб для класу N_samples можна було створити максимум 7 екземлярів.
6. Які відмінності між методами __new__ та __init__?
7. Чи можна змінити статичний атрибут класу через ім’я класу?
8. Чи можна змінити динамічний атрибут класу через ім’я класу?
9. Поясніть принципи як з застосуванням класів здійснюється реалізація поліморфізму та спадкування.

РОЗДІЛ 2. Використання мови програмування python для аналізу експериментальних даних

2.1. Пакет numpy мови програмування python

Пакет numpy (numeric python) – це бібліотека на мові python, що допомагає підтримувати більшість різноманітних масивів і матриць, разом з великою бібліотекою високорівневих (і дуже швидких) математичних функцій для операцій з цими масивами [21, 23]. Офіційний сайт бібліотеки numpy – www.numpy.org [24].

Пакет numpy означає числовий python (numeric python). Це бібліотека, що складається з об'єктів багатовимірного масиву та набору процедур обробки масиву. Numeric, попередник пакету numpy, був розроблений Джимом Хугуніним. Пізніше було розроблено ще один пакет numarray (numeric array), який мав додаткові функції. В 2005 Тревіс Оліфант створив пакет numpy, включивши функції numarray в пакет numeric. В розробці цього пакету брали участь багато учасників проєкту, так як він має відкритий вихідний код.

Використовуючи пакет numpy, розробник може виконувати такі операції:

- Математичні та логічні операції над масивами, потужний N-мірний масив.
- Високорівневі функції.
- Інструменти для інтеграції коду мов програмування C/C++ та Fortran.
- Використання лінійної алгебри, перетворень Фур'є та роботи з випадковими числами.

Пакет numpy – досить низькорівневий інструмент, схожий на MATLAB.

Пакет numpy використовується разом з такими пакетами, як scipy (scientific python) [25] та matplotlib (бібліотека для побудови графіків та діаграм) [26]. Комбінація цих пакетів широко використовується як заміна MatLab, популярної платформи для технічних обчислень. Тим не менш, python як альтернатива MatLab тепер розглядається як більш сучасна та повна мова програмування.

Стандартна установка python не постачається з пакетом numpy. Необхідно встановлювати numpy за допомогою вбудованого пакетного менеджера pip або за допомогою розділу python packages у PyCharm. Для повноцінного аналізу експериментальних даних в python необхідно встановити пакети і numpy і scipy, останній побудовано на базі numpy. Пакет numpy містить дані про масиви та операції, такі як сортування, індексація. Але хоча в numpy є функції для роботи з лінійною алгеброю, у scipy вони представлені у повному обсязі разом з масою інших функцій.

2.1.1. Створення масивів в пакеті numpy

У програмуванні масив (англ. array) – сукупність елементів одного типу даних, впорядкованих за індексами, що визначають положення елемента в масиві. Масив може бути одновимірним (вектором), та багатовимірним (наприклад, таблицею), тобто таким, де індексом є не одне число, а кортеж (сукупність) з декількох чисел, кількість яких збігається з розмірністю масиву.

Пакет numpy має інструменти роботи з багатовимірними масивами даних у формі рядків та стовпчиків. Масив numpy – це не те саме, що клас array.array зі стандартної бібліотеки python, який працює тільки з одновимірними масивами. За допомогою методу array() пакету numpy можна список даних перетворити в масив даних [27] (Рисунок 2.1).

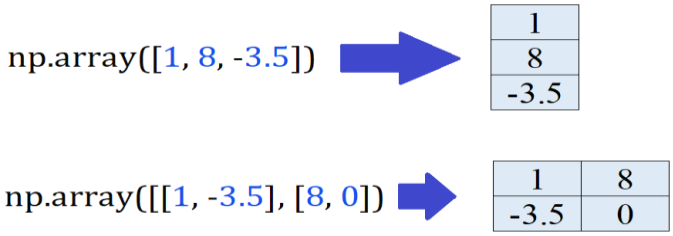


Рисунок 2.1 – Схема застосування методу array() пакету numpy.

Для використання функціоналу пакету numpy його потрібно імпортувати до файлу. Загальноприйнятим є застосування псевдоніму – np, що дозволяє лаконічніше записувати звернення до вмісту пакету numpy. Щоб додати псевдонім достатньо після імпорту додати прислівник as та нове ім'я, тоді замість вказання повної назви достатньо використати np:

```
import numpy as np

a = np.array([1, 2, 3])
```

Спочатку нам потрібно імпортувати пакет numpy під псевдонімом np, всі об'єкти створені в коді, які належать пакету numpy будуть починатись з "`np`".

Масиви можна створювати багатьма способами:

- зі списку за використанням методу пакету numpy np.array()
- з використанням методу пакету numpy np.arange(n) - цілі числа від 0 включно до n, крім n
- з використанням методу пакету numpy np.linspace(start, end, n) - рівномірно розміщених на проміжку [start, end] n чисел
- з використанням функцій тощо.

Якщо тип dtype таких масивів не вказано, за замовчуванням буде використовуватися dtype = int32, float64, complex128.

Приклад коду для створення масивів numpy – *numpy*.array([list]), numpy.arrange(start, stop, step), numpy.linspace(start, stop, num of elements):

```
import numpy as np
print('numpy.array([list]) – функція, яка створює масив numpy зі
списку list')
 # створюємо обєкт a – одновимірний масив numpy
a = np.array([1,2,3])
print('a = \n', a)
# створюємо обєкт a1 – двомірний масив numpy
a1 = np.array([[1,2,3,4],[5,6,7,8]])
print('a1 = \n', a1, '\n')

print('numpy.arrange(start, stop, step) – функція, яка створює
масив numpy, \n'
      'елементи якого лежать у діапазоні значень від start до stop
з кроком step')
a2 = np.arange(4)         # цілі числа від 0 включно до n,
print(f'a2 = np.arange(4) = {a2}')
a21 = np.arange(5,26,5)
print(f'a21 = np.arange(5,26,5) {a21} \n')

print('numpy.linspace(start, stop, num_of_elements) – функція, яка
```

```
створює масив numpy, елементи якого \n'
      'лежать у діапазоні значень між start до stop,
num_of_elements – кількість елементів у  масиві')
a3 = np.linspace(-10, 10, 5)  # 5 рівномірно розміщених на
проміжку [-10, 10] чисел
print(f'a3 = np.linspace(-10, 10, 5) \n{a3 = }')
a31 = np.linspace(-10, 10)  #  рівномірно розміщених на проміжку
[-10, 10] чисел
print(f'a31 = np.linspace(-10, 10) - \n{a31 = }')
a32 = np.linspace(1, 10, dtype=int)  #  50 рівномірно розміщених
на проміжку [1, 10] чисел
print(f'a32 = np.linspace(1, 10, dtype=int) - \n{a32 = } \n')

Результат роботи коду:
numpy.array([list]) – функція, яка створює масив numpy зі списку
list
a =
 [1 2 3]
a1 =
 [[1 2 3 4]
 [5 6 7 8]]
numpy.arrange(start, stop, step) – функція, яка створює масив
numpy,
елементи якого лежать у діапазоні значень від start до stop з
кроком step
a2 = np.arange(4) = [0 1 2 3]
a21 = np.arange(5,26,5) [ 5 10 15 20 25]
numpy.linspace(start, stop, num_of_elements) – функція, яка
створює масив numpy, елементи якого
лежать у діапазоні значень між start до stop, num_of_elements –
кількість елементів у  масиві
a3 = np.linspace(-10, 10, 5)
a3 = array([-10.,  -5.,   0.,   5.,  10.])
a31 = np.linspace(-10, 10) -
a31 = array([-10.        ,  -9.59183673,  -9.18367347,  -8.7755102
,
        -8.36734694,  -7.95918367,  -7.55102041,  -7.14285714,
        -6.73469388,  -6.32653061,  -5.91836735,  -5.51020408,
        -5.10204082,  -4.69387755,  -4.28571429,  -3.87755102,
        -3.46938776,  -3.06122449,  -2.65306122,  -2.24489796,
        -1.83673469,  -1.42857143,  -1.02040816,  -0.6122449 ,
        -0.20408163,   0.20408163,   0.6122449 ,   1.02040816,
         1.42857143,   1.83673469,   2.24489796,   2.65306122,
         3.06122449,   3.46938776,   3.87755102,   4.28571429,
         4.69387755,   5.10204082,   5.51020408,   5.91836735,
         6.32653061,   6.73469388,   7.14285714,   7.55102041,
         7.95918367,   8.36734694,   8.7755102 ,   9.18367347,
         9.59183673,  10.        ])
 a32 = np.linspace(1, 10, dtype=int)
a32 = array([ 1,  1,  1,  1,  1,  1,  2,  2,  2,  2,  2,  3,  3,
3,  3,  3,  3,
        4,  4,  4,  4,  4,  5,  5,  5,  5,  5,  5,  6,  6,  6,  6,
6,  7,
```

```
        7,  7,  7,  7,  7,  8,  8,  8,  8,  8,  9,  9,  9,  9,  9,
10])
```

В пакеті numpy є функції (методи), які створюють нові масиви заданої розмірності, де кожен елемент генерується випадковим чином, заповнюється нулями, заповнюється одиницями або вміст якого є довільним і залежить від стану пам'яті (Рисунок 2).
Приклад коду по створенню масивів – numpy.random.random((rows, column)):

```
import numpy as np
print('numpy.random.random((rows, column)) – повертає масив
заданої розмірності, \n'
      'де кожен елемент генерується випадковим чином')
a1 = np.random.random((3, 2))
print('a1(3,2) =  \n', a1)
a2 = np.random.random((2, 3))
print('a2(2,3) =  \n', a2)
   Результат роботи коду:
a1(3,2)
 [[0.96827682 0.67088983]
  [0.96176924 0.35060824]
  [0.82134705 0.71944234]]
a2(2,3)
 [[0.58716665 0.81637361 0.37472359]
  [0.97108291  0.93343907 0.71469363]]
```

Оскільки в методі numpy.random.random((rows, column)) використовується генератор випадкових чисел, результати при кожному виконанні коду будуть відрізнятися.

Приклад коду по створенню масивів – numpy.zeros((rows, columns)) [27], numpy.ones((rows, columns)) [27], numpy.empty((rows, columns)) (Рисунок 2.2):

```
import numpy as np
print('numpy.zeros((rows, columns)) створює масив з заданою
розмірністю, де кожен елемент дорівнюватиме 0')
print('numpy.ones((rows, columns)) створює масив з заданою
розмірністю, де кожен елемент дорівнюватиме 1')
print("numpy.empty((rows, columns)) створює масив, вміст якого є
випадковим і залежить від стану пам'яті \n"
      "тобто масив не ініціалізованих (empty - довільних) даних з
заданими атрибутами")
a_zeros = np.zeros((3,3), dtype=float)
print('a_zeros = ', a_zeros)
a_ones = np.ones((3,3), dtype=int)
print('a_ones = ', a_ones)
a_empty = np.empty((3,3))
print('a_empty = ', a_empty, '\n')
   Результат роботи коду:
numpy.zeros((rows, columns)) створює масив з заданою розмірністю,
де кожен елемент дорівнюватиме 0
numpy.ones((rows, columns)) створює масив з заданою розмірністю,
де кожен елемент дорівнюватиме 1
numpy.empty((rows, columns)) створює масив, вміст якого є
випадковим і залежить від стану пам'яті
```

```
тобто масив не ініціалізованих (empty - довільних) даних з
заданими атрибутами
a_zeros =
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]


a_ones =
 [[1 1 1]
 [1 1 1]
 [1 1 1]]
a_empty =
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

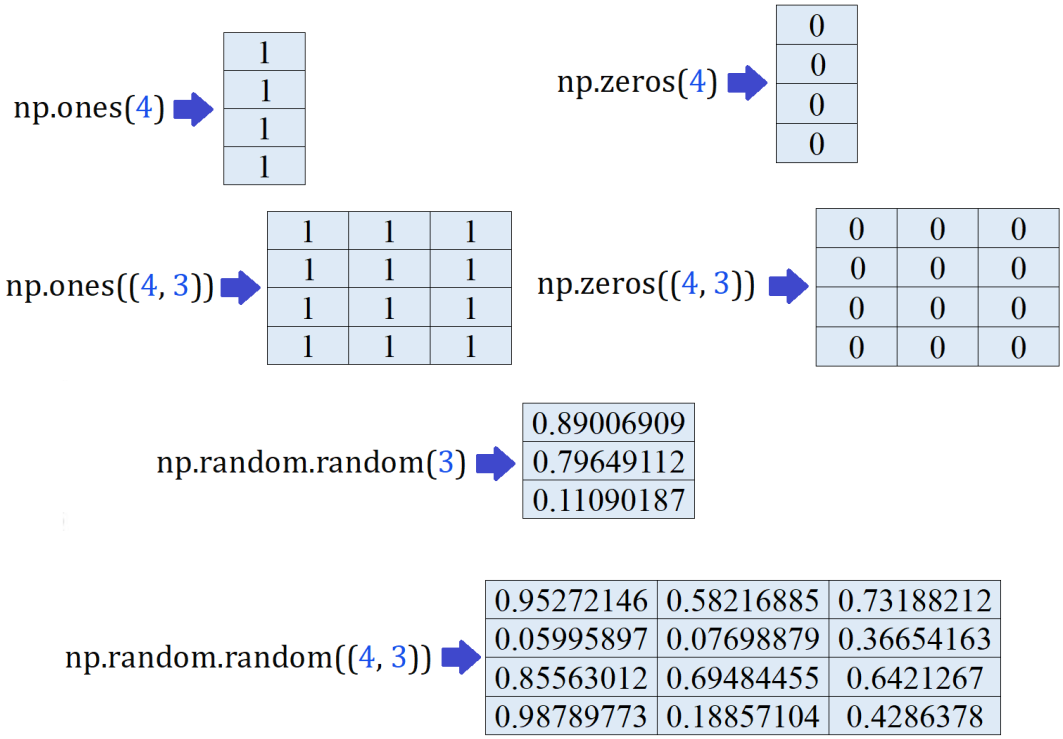


Рисунок 2.2 – Ілюстрація функцій numpy: numpy.ones, numpy.zeros, numpy.empty, numpy.random.random

2.1.2. Слайси (зрізи), індексування, ітерації масивів numpy

Індексацію масивів схематично представлено на Рисунок 2.3, індекси приймають цілочисельні значення, які починаються з нуля, при індексуванні багатовимірного масиву індекси розділяють комами. Як і зі списків, з масивів можна робити слайси та ітерацію елементів масиву, як в наступному прикладі:

```
import numpy as np
list1 = [0, 1, 2, 3]
print('маємо список - list1 = [0, 1, 2, 3]',)
a = np.array(list1)
```

```
print(f'створюємо масив a зі списку list1, a = np.array(list1): {a
= }')
print(f'слайс з масиву np.array(a[:-1]) = - {np.array(a[:-1])}')

a1 = np.array([[1, 2, 3, 4,], [11, 12, 13, 14,]])
print(f'a1 = {a1}')
print(f'друк елементу масиву a1[1,2] = {a1[1,2]}')
```

Результат роботи коду:

```
маємо список - list1 = [0, 1, 2, 3]
створюємо масив a зі списку list1, a = np.array(list1): a =
array([0, 1, 2, 3])
слайс з масиву np.array(a[:-1]) = [0 1 2]
a1 = [[ 1  2  3  4]
 [11 12 13 14]]
друк елементу масиву a1[1,2] = 13
```

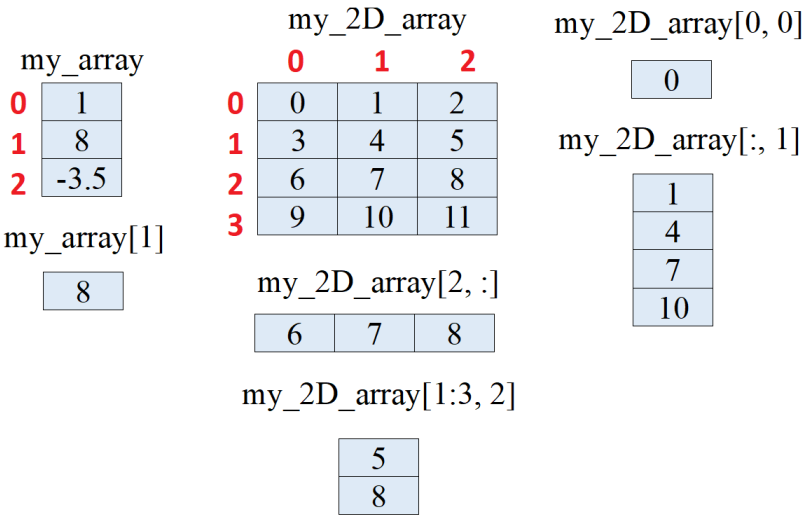


Рисунок 2.3 – Ілюстрація індексації масивів numpy.

2.1.3. Атрибути (поля) масиву numpy

Клас в якому зберігаються масиви називається `ndarray`, і він має такі атрибути (поля):

- ndarray.dtype - об'єкт що описує тип елементів масиву, наприклад, заданий в numpy: bool, character, int, int8, int16, int32, int64, float, float8, float16, float32, float64, complex, complex64, object
- ndarray.data – адреса буферу в пам'яті комп'ютера, який займає даний масив
- ndarray.ndim – розмірність масиву
- ndarray.size - загальну кількість елементів у масиві (дорівнює добутку всіх чисел в shape)
- ndarray.itemsize – довжина кожного елемента масиву в байтах
- ndarray.shape – форма масиву, повертає кортеж що зберігає розмір вздовж кожного виміру масиву.

Приклад коду по визначенню атрибутів масиву – ndarray.dtype, ndarray.data:

```
import numpy as np
print('type() - вбудована функція python, повертає тип переданого
параметра, тобто тип масиву')
```

```
print("ndarray.data – повертає адресу буферу в пам'яті комп'ютера,
який займає даний масив")
print("ndarray.dtype – повертає об'єкт, який описує тип елементів
у масиві")
a = np.array([1,2,3])
print('a = ', a)
print(f'type(a) = {type(a)}')
print('a.data = ', a.data, '\n')
a = np.array([1,2,3])
print('Тип елементів масиву a = np.array([1,2,3]):')
print('тип не задано явно: np.array([1,2,3]), a.dtype =  ',
a.dtype)
a = np.array([1,2,3], dtype=int)
print('np.array([1,2,3], dtype=int), a.dtype =  ', a.dtype)
print('a = ', a)
a = np.array([1,2,3], dtype=float)
print('np.array([1,2,3], dtype=float), a.dtype = ', a.dtype)
print('a = ', a)
a = np.array([1,2,3], dtype=complex)
print('np.array([1,2,3], dtype=float), a.dtype = ', a.dtype)
print('a = ', a, '\n')
```

Результат роботи коду:

```
type() - вбудована функція python, повертає тип переданого
параметра, тобто тип масиву
ndarray.data – повертає адресу буферу в пам'яті комп'ютера, який
займає даний масив
ndarray.dtype – повертає об'єкт, який описує тип елементів у
масиві
a =  [1 2 3]
type(a) = <class 'numpy.ndarray'>
a.data =  <memory at 0x000001DD3DF55880>
Тип елементів масиву a = np.array([1, 2, 3]):
тип не задано явно: np.array([1,2,3]), a.dtype  =  int32
np.array([1,2,3], dtype=int), a.dtype =   int32
a =  [1 2 3]
np.array([1,2,3], dtype=float), a.dtype =  float64
a =  [1. 2. 3.]
np.array([1,2,3], dtype=float), a.dtype =  complex128
a =  [1.+0.j 2.+0.j 3.+0.j]
```

Приклад коду по визначенню атрибутів масиву numpy – ndarray.ndim, ndarray.size:

```
import numpy as np
print('ndarray.ndim повертає розмірність масиву')
print('ndarray.size – повертає загальну кількість елементів у
масиві')
a = np.array([1, 2, 3])
print('a = \n', a)
print('a.ndim = ', a.ndim)
print('a.size = ', a.size)
a1 = np.array([[1, 2, 3]])
print('a1 = \n', a1)
print('a1.ndim = ', a1.ndim)
```

```
print('a1.size = ', a1.size)
a2 = np.array([['a', 'b', 'c'], [4, 5, 6], [7, 'c', 8]])
print('a2 = \n', a2)
print('a2.ndim = ', a2.ndim)
print('a2.size = ', a2.size)
a3 = np.array([[['a', 'b', 'c'], [4, 5, 6], [7, 'c', 8]]])
print('a3 = \n', a3)
print('a3.ndim = ', a3.ndim)
print('a3.size = ', a3.size)
```

Результат роботи коду:

```
ndarray.ndim повертає розмірність масиву
ndarray.size - повертає загальну кількість елементів у масиві
a =
 [1 2 3]
a.ndim =  1
a.size =  3
a1 =
 [[1 2 3]]
a1.ndim =  2
a1.size =  3
a2 =
 [['a' 'b' 'c']
 ['4' '5' '6']
 ['7' 'c' '8']]
a2.ndim =  2
a2.size =  9
a3 =
 [[['a' 'b' 'c']
  ['4' '5' '6']
  ['7' 'c' '8']]]
a3.ndim =  3
a3.size =  9
```

Приклад коду по визначенню атрибуту масиву numpy – ndarray.itemsize:

```
import numpy as np
print('ndarray.itemsize - повертає довжину кожного елемента масиву
в байтах')
a = np.array([[1, 2, 3],[4, 5 ,6]], dtype=np.int32)
print('a = \n', a)
print("довжина кожного елемента масиву a в байтах
(dtype=np.int32): ", a.itemsize)
a1 = np.array([[1, 2, 3],[4, 5 ,6]], dtype=np.float64)
print('a1 = \n', a1)
print("довжина кожного елемента масиву a1 в байтах
(dtype=np.float64): ", a1.itemsize)
a2 = np.array([[1, 2, 3],[4, 5 ,6]], dtype=np.complex128)
print('a2 = \n', a2)
print("довжина кожного елемента масиву a2 в байтах
(dtype=np.complex128): ", a2.itemsize)
```

Результат роботи коду:

```
ndarray.itemsize - повертає довжину кожного елемента масиву в
байтах
```

```
a =
 [[1 2 3]
 [4 5 6]]
довжина кожного елемента масиву a в байтах (dtype=np.int32):  4
a1 =
 [[1. 2. 3.]
 [4. 5. 6.]]
довжина кожного елемента масиву a1 в байтах (dtype=np.float64):  8
a2 =
 [[1.+0.j 2.+0.j 3.+0.j]
 [4.+0.j 5.+0.j 6.+0.j]]
довжина кожного елемента масиву a2 в байтах (dtype=np.complex128):
16
```

Масиви в numpy можуть бути багатовимірними, розмірність задається атрибутом `shape, який` повертає кортеж форми масиву.

Форма масиву в numpy задається атрибутом ndarray.shape:

- для одновимірного масиву ndarray.shape повертає кортеж (m,), де m – кількість елементів у масиві;
- для двомірного масиву ndarray.shape повертає кортеж (n, m), де n – кількість рядків, а m – кількість стовпчиків;
- для тримірного масиву ndarray.shape повертає кортеж (k, n, m), де k - глибина масиву, n – кількість рядків, m – кількість стовпчиків.

Приклад коду по визначенню атрибуту масиву numpy – ndarray.shape

```
import numpy as np
print('ndarray.shape повертає кортеж розмірностей масиву: (m,),
(n, m), (k, n, m)')
a0 = np.array([1, 2, 3])
print(f'{a0 = } {a0.shape = }')
a1 = np.array([[1, 2, 3], [4, 5, 6]])
print(f'{a1 = } {a1.shape = }')
a2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 6]])
print(f'{a2 = } {a2.shape = }')
a3 = np.array([[[1, 2, 3], [4, 5, 6], [7, 8, 6]]])
print(f'{a3 = } {a3.shape = } \n')
```

Результат роботи коду:

```
ndarray.shape повертає кортеж розмірностей масиву: (m,), (n, m),
(k, n, m)
a0 = array([1, 2, 3])  a0.shape = (3,)
a1 = array([[1, 2, 3],
       [4, 5, 6]])            a1.shape = (2, 3)
a2 = array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 6]])           a2.shape = (3, 3)
a3 = array([[[1, 2, 3],
        [4, 5, 6],
        [7, 8, 6]]])          a3.shape = (1, 3, 3)
```

2.1.4. Базові арифметичні операції з масивами в numpy

Базові арифметичні операції (додавання, віднімання, множення, піднесення до степеня, ділення, цілочисельне ділення, ділення по модулю) з масивами в numpy виконуються поелементно, тобто створюється новий масив (Рисунок 2.4), в який і записується результат.

Приклад коду арифметичних операцій з масивами в numpy:

```
import numpy as np
a = np.array([5, 10, 15, 20, 25])
print('a', a)
b = np.array([1, 2, 3, 6, 8])
print('b', b)
c = a + b                  # повертає добуток елементів двох
масивів;
print('a + b ', a + b)
a - b                      # повертає різницю елементів двох
масивів;
print('a - b', a - b )
a*b                        # повертає добуток елементів обох
масивів.
print('a*b ', a*b)
a**b                       # піднесення елементів масиву a до
степеня відповідного елементу масиву b
print('a**b', a**b)
b**3                      # повертає масив, де кожне значення
піднесено до квадрату;
print('a**2', a**2)
a/b                       # повертає частку від ділення елементів
масиву a на відповідний елемент масиву b
print('a/b', a/b)
a/2                       # повертає частку від ділення елементів
масиву a на задане число
print('a/2', a/2)
a//b                      # повертає неповну частку від ділення
елементів масиву a на відповідний елемент масиву b для найближчого
найменшого цілого
print('a//b', a//b)
a//3                      # повертає неповну частку від ділення
елементів масиву a на задане число  для найближчого найменшого
цілого
print('a//3', a//3)
a%b                       # повертає залишок від ділення елементів
масиву a на відповідний елемент масиву b
print('a%b', a%b)
a%2                       # повертає залишок від ділення елементів
масиву a на задане число
print('a%2', a%2)
10*np.sin(a)              # повертає значення відповідно до заданого
виразу
```

Результат роботи коду:

```
a [ 5 10 15 20 25]
b [1 2 3 6 8]
a + b  [ 6 12 18 26 33]
a - b [ 4  8 12 14 17]
```

```
a*b  [  5  20  45 120 200]
a**b [          5         100        3375    64000000 -2030932031]
a**2 [ 25 100 225 400 625]
a/b [5.          5.          5.          3.33333333 3.125     ]
a/2 [ 2.5  5.   7.5 10.  12.5]
a//b [5 5 5 3 3]
a//3 [1 3 5 6 8]
a%b [0 0 0 2 1]
a%2 [1 0 1 0 1]
```

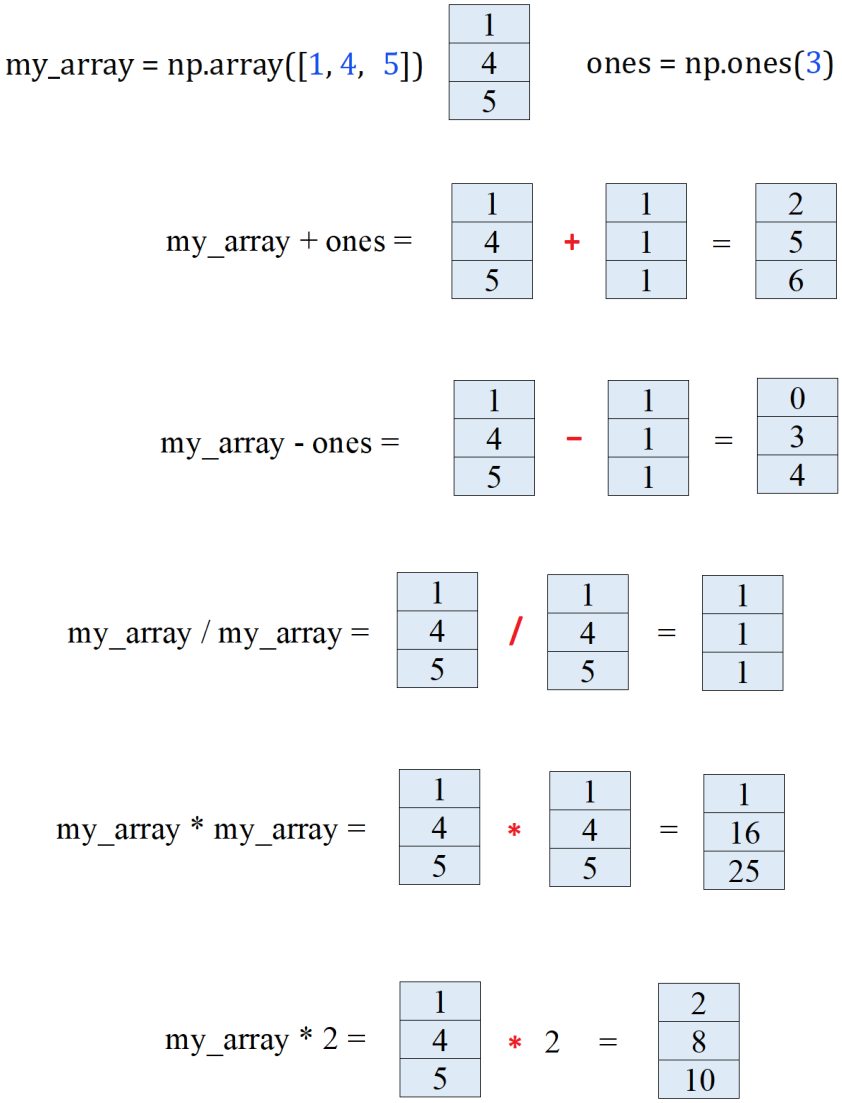


Рисунок 2.4 – Ілюстрація базових арифметичних операцій з масивами однакових розмірів numpy.

2.1.5. Основні методи роботи з масивами в numpy

Основні методи (функції) роботи з масивами в пакеті numpy наведено нижче:
numpy.ndarray.sum() повертає суму всіх елементів масиву
ndarray.min() – функція, яка повертає елемент із мінімальним значенням даних з масиву
ndarray.max() – функція, яка повертає елемент із максимальним значенням даних з масиву
numpy.around(a, decimals = 0) – функція округляє a до заданої кількості десяткових розрядів, за замовчуванням до цілого. Дотримується такого правила округлення: якщо значення a знаходиться точно посередині між двома цілими, округлення проводитися до найближчого

парного цілого. Аргумент decimals – ціле число, десятковий розряд після коми, до якого проводиться округлення, якщо decimals негативне, розряд відраховується ліворуч від коми.
numpy.fix(a) – відкидає дробову частину a;
numpy.ceil(a) – округляє a до більшого з цілих;
numpy.floor(a) – округляє a до меншого з цілих;
numpy.sign(a) – повертає -1 якщо a < 0; повертає 0, якщо a == 0 та повертає 1 якщо a > 0;
numpy.exp(a) – повертає основу натурального логарифму (число e) у степені a
numpy.exp(numpy.ndarray) – функція поверне масив ndarray з експонентою від величини кожного елемента масиву
numpy.log(a), numpy.log10(a), numpy.log2(a) - повертає натуральний логарифм a, логарифм a за основою 10, логарифм a за основою 2
numpy.sqrt(numpy.ndarray) – функція поверне масив ndarray з квадратним коренем кожного елемента
numpy.abs(a) – функція повертає абсолютне значення a
numpy.cos(a), numpy.sin(a), numpy.tan(a) – повертає косинус, синус, тангенс a у радіанах
numpy.cosh(a), numpy.sinh(a), numpy.tanh(a) – повертає гіперболічний косинус, синус, тангенс a
numpy.arccos(a), numpy.arcsin(a), numpy.arctan(a) – повертає арккосинус, арксинус, арктангенс a в радіанах, для арккосинусу в діапазоні [0, nump.pi], для арксинуса - [-numpy.pi/2, numpy.pi/2], для арктангенса - [-numpy.pi/2, numpy.pi/2]
numpy.arccosh(a), numpy.arcsinh(a), numpy.arctanh(a) – повертає гіперболічний арккосинус, арксинус, арктангенс a
numpy.degrees(a) – конвертує a з радіан у градуси;
numpy.radians(a) – конвертує a із градусів у радіани;
numpy.reshape(dimensions) – ця функція використовується для зміни розмірності масиву numpy. Від кількості аргументів у reshape залежить, яку розмірність буде мати масив numpy.

Приклад коду методів numpy (Рисунок 2.5) – ndarray.sum(), ndarray.min(), ndarray.max():

```
import numpy as np
print('ndarray.sum() повертає суму всіх елементів масиву')
print('ndarray.min() – функція, яка повертає елемент із
мінімальним значенням даних з масиву')
print('ndarray.max() – функція, яка повертає елемент із
максимальним значенням даних з масиву')
a = np.array([[1, 2, 3], [4, 5, 6]])
print('a = \n', a)
a_random = np.random.random((2, 3))
print('a_random  = \n', a_random)
print('a.sum()) = ', a.sum())
print('a_random.sum() = ', a_random.sum())
print('a.min() = ', a.min())
print('a.max() = ', a.max())
print('a_random.min() = ',  a_random.min())
print('a_random.max() = ', a_random.max())
```

Результат виконання коду:

```
ndarray.sum() повертає суму всіх елементів масиву
ndarray.min() – функція, яка повертає елемент із мінімальним
значенням даних з масиву
ndarray.max() – функція, яка повертає елемент із максимальним
значенням даних з масиву
a =
 [[1 2 3]
```

```
 [4 5 6]]
a_random  =
 [[0.95122379 0.25737903 0.81831147]
 [0.45279409 0.42550206 0.55193893]]
a.sum()) =  21
a_random.sum() =  3.457149368110653
a.min() =  1
a.max() =  6
a_random.min() =  0.25737902684318337
a_random.max() =  0.9512237914884767
```

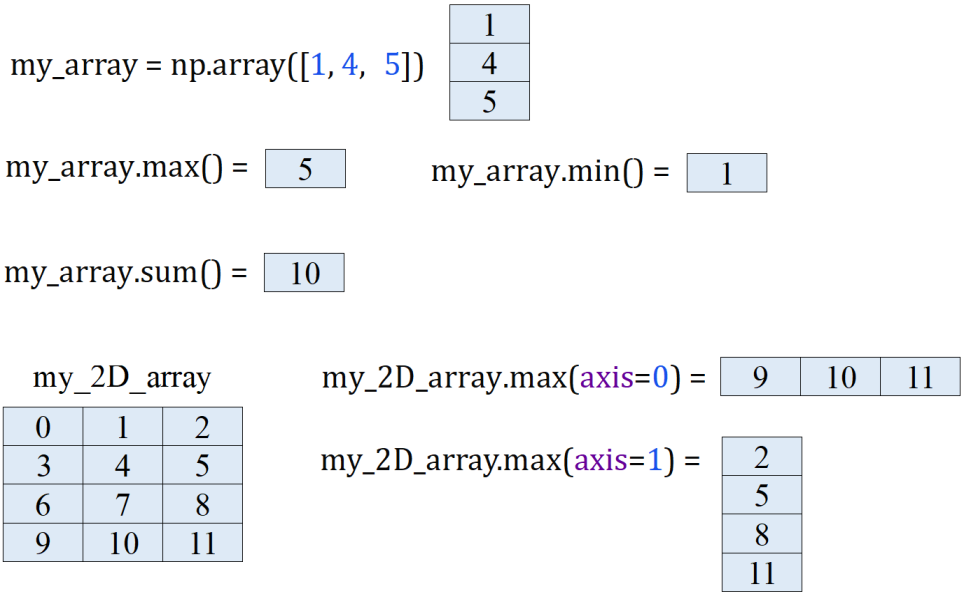


Рисунок 2.5 – Ілюстрація застосування функцій ndarray.sum(), ndarray.min(), ndarray.max() з масивами numpy

Приклад методів numpy – numpy.around(a), numpy.fix(a), numpy.ceil(a), numpy.floor(a), numpy.sign(a):
numpy.around(a, decimals = 0) – функція округляє a до заданої кількості десяткових розрядів, за замовчуванням до цілого. Дотримується такого правила округлення. Якщо значення a знаходиться точно посередині між двома цілими, округлення проводитися до найближчого парного цілого.

Аргумент decimals – ціле число, десятковий розряд після коми, до якого проводиться округлення, якщо decimals негативне, розряд відраховується ліворуч від коми.

```
import numpy as np
print('numpy.around(a, decimals) – округляє a до заданої кількості
десяткових розрядів, за замовчуванням до цілого.\n '
      'Дотримується такого правила округлення: якщо значення a
знаходиться точно посередині між двома цілими,\n '
      'округлення проводитися до найближчого парного цілого.
Аргумент decimals – ціле число, десятковий розряд\n '
      'після коми, до якого проводиться округлення, якщо decimals
негативне, розряд відраховується ліворуч від коми')
print('numpy.fix(a) – відкидає дробову частину')
print('numpy.ceil(a) – округляє a до більшого з цілих')
print('numpy.floor(a) – округляє a до меншого з цілих')
print('numpy.sign(a) – повертає  -1 якщо a < 0; повертає 0, якщо a
```

```
== 0 та повертає 1 якщо a > 0')
a = 90.875
print('a = ', a)
print(f'np.fix(a) = {np.fix(a)}')
print(f'np.ceil(a) = {np.ceil(a)}')
print(f'np.floor(a) = {np.floor(a)}')
print(f'np.degrees(np.pi) = {np.degrees(np.pi)}')
print(f'np.radians(180) = {np.radians(180)}')
print(f'np.sign(a) = {np.sign(a)}')
print(f'np.sign(-a) = {np.sign(-a)}')
print(f'a < 15 = {a < 15}')
```

Результат роботи коду:

```
numpy.around(a, decimals) - округляє a до заданої кількості
десяткових розрядів, за замовчуванням до цілого.
 Дотримується такого правила округлення: якщо значення a
знаходиться точно посередині між двома цілими,
 округлення проводитися до найближчого парного цілого. Аргумент
decimals - ціле число, десятковий розряд
 після коми, до якого проводиться округлення, якщо decimals
негативне, розряд відраховується ліворуч від коми
numpy.fix(a) - відкидає дробову частину
numpy.ceil(a) - округляє a до більшого з цілих
numpy.floor(a) - округляє a до меншого з цілих
numpy.sign(a) - повертає  -1 якщо a < 0; повертає 0, якщо a == 0
та повертає 1 якщо a > 0
a =  90.875
np.fix(a) = 90.0
np.ceil(a) = 91.0
np.floor(a) = 90.0
np.degrees(np.pi) = 180.0
np.radians(180) = 3.141592653589793
np.sign(a) = 1.0
np.sign(-a) = -1.0
a < 15 = False
```

Приклад коду методів numpy – numpy.exp(a), numpy.exp(numpy.ndarray), numpy.log(a), numpy.log10(a), numpy.log2(a), numpy.sqrt(numpy.ndarray), numpy.abs(a):

```
import numpy as np
print('np.exp(a) - повертає основу натурального логарифму (число
e) у степені a')
print('np.exp(numpy.ndarray) - функція поверне масив ndarray з
експонентою від величини кожного елемента масиву')
print('np.log(a), np.log10(a), np.log2(a) - натуральний логарифм
a, логарифм a за основою 10, логарифм a за основою 2')
print('np.sqrt(numpy.ndarray) - функція повертає масив ndarray з
квадратним коренем кожного елемента')
print('np.abs(a) - функція повертає абсолютне значення a\n')

print(f'np.exp([10]) = {np.exp([10])}')          # число e в
степені 10
print(f'{np.e ** 10 = }')
a = np.array([1, 2, 3])
```

```
print(f'{a = }')
print(f'{np.exp(a) = }')
print(f'{np.log(a) = }')
print(f'{np.log10(a) = }')
print(f'{np.log2(a) = }')
print(f'{np.sqrt([16, 4, 256, 120]) = }')
print(f'{np.abs(a) = }')
b = np.array([-1, -2, 0.3])
print(f'{np.abs(b) = }')
```

Результат роботи коду:

```
np.exp(a) – повертає основу натурального логарифму (число e) у
степені a
np.exp(numpy.ndarray) – функція поверне масив ndarray з
експонентою від величини кожного елемента масиву
np.log(a), np.log10(a), np.log2(a) - натуральний логарифм a,
логарифм a за основою 10, логарифм a за основою 2
np.sqrt(numpy.ndarray) – функція повертає масив ndarray з
квадратним коренем кожного елемента
np.abs(a) – функція повертає абсолютне значення a
np.exp([10]) = [22026.46579481]
np.e ** 10 = 22026.465794806703
a = array([1, 2, 3])
np.exp(a) = array([ 2.71828183,  7.3890561 , 20.08553692])
np.log(a) = array([0.        , 0.69314718, 1.09861229])
np.log10(a) = array([0.        , 0.30103   , 0.47712125])
np.log2(a) = array([0.       , 1.       , 1.5849625])
np.sqrt([16, 4, 256, 120]) = array([ 4.        ,  2.        , 16.
, 10.95445115])
np.abs(a) = array([1, 2, 3])
np.abs(b) = array([1. , 2. , 0.3])
```

Приклад коду тригонометричних функцій – numpy.sin(a), numpy.cos(a), numpy.tan(a), numpy.arcsin(a), numpy.arccos(a), numpy.arctan(a) *і гіперболічних функцій* – numpy.cosh(a), numpy.sinh(a), numpy.tanh(a), numpy.arccosh(a), numpy.arcsinh(a), numpy.arctanh(a) (Рисунок 2.6):

```
import numpy as np
a = np.array([0, 1, np.pi/2, np.pi, 2*np.pi])
print(f'{a = }')
print('**тригонометричні функції numpy')
print(f'{np.around(np.sin(a), 2) = }')
print(f'{np.around(np.cos(a), 2) = }')
print(f'{np.around(np.tan(a), 2) = }, tan(a) = sin(a) / cos(a)')
print(f'{np.arcsin(a) = }')
print(f'{np.arccos(a) = }')
print(f'{np.arctanh(a) = }')
print('**гіперболічні функції numpy')
print(f'{np.sinh(a) = }')
print(f'{np.cosh(a) = }')
print(f'{np.tanh(a) = }')
print(f'{np.arcsinh(a) = }')
print(f'{np.arccosh(a) = }')
print(f'{np.arctanh(a) = }')
```

```
   Результат роботи коду:
a = array([0.        , 1.        , 1.57079633, 3.14159265,
6.28318531])
**тригонометричні функції numpy
np.around(np.sin(a), 2) = array([ 0.  ,  0.84,  1.  ,  0.  , -0.
])
np.around(np.cos(a), 2) = array([ 1.  ,  0.54,  0.  , -1.  ,  1.
])
np.around(np.tan(a), 2) = array([ 0.00000000e+00,  1.56000000e+00,
1.63312394e+16, -0.00000000e+00,
       -0.00000000e+00]), tan(a) = sin(a) / cos(a)
np.around(np.arcsin(a), 2) = array([0.  , 1.57,  nan,  nan,  nan])
np.around(np.arccos(a), 2) = array([1.57, 0.  ,  nan,  nan,  nan])
np.around(np.arctanh(a), 2) = array([ 0., inf, nan, nan, nan])
**гіперболічні функції numpy
np.around(np.sinh(a), 2) = array([  0.  ,   1.18,   2.3 ,  11.55,
267.74])
np.around(np.cosh(a), 2) = array([  1.  ,   1.54,   2.51,  11.59,
267.75])
np.around(np.tanh(a), 2) = array([0.  , 0.76, 0.92, 1.  , 1.  ])
np.around(np.arcsinh(a), 2) = array([0.  , 0.88, 1.23, 1.86,
2.54])
np.around(np.arccosh(a), 2) = array([ nan, 0.  , 1.02, 1.81,
2.52])
np.around(np.arctanh(a), 2) = array([ 0., inf, nan, nan, nan])
```

Приклад коду функції numpy.reshape(dimensions) (Рисунок 2.6):

```
import numpy as np
print('numpy.reshape(dimensions + i) – функція використовується
для зміни розмірності масиву,\n'
'розмірність масиву визначається кількістю аргументів функції
np.reshape(n,m) або np.reshape(k,n,m)\n'
      'i - індекс символа, з якого починається масив, по
замовчуванню i = 0')
a = np.array([1, 2, 3, 4, 5, 6, 7, 8])
print(f'a = {a}')
print(f'a.reshape(2, 4) = \n{a.reshape(2, 4)}')
a = np.arange(6).reshape(2, 3)
print(f'a = np.arange(6).reshape(2,3) = \n{a}')
a = np.arange(6).reshape(2, 3) + 3
print(f'a = np.arange(6).reshape(2,3) + 3 = \n{a}')
a = np.arange(12).reshape(2, 2, 3)
print(f'a = np.arange(12).reshape(2,3) = \n{a}\n')
```

```
   Результат роботи коду:
numpy.reshape(dimensions + i) – функція використовується для зміни
розмірності масиву, розмірність масиву визначається кількістю
аргументів функції np.reshape(n,m) або np.reshape(k,n,m); i -
індекс символа, з якого починається масив, по замовчуванню i = 0.
a = [1 2 3 4 5 6 7 8]
a.reshape(2, 4) =
[[1 2 3 4]
 [5 6 7 8]]
```

```
a = np.arange(6).reshape(2,3) =
[[0 1 2]
 [3 4 5]]
a = np.arange(6).reshape(2,3) + 3 =
[[3 4 5]
 [6 7 8]]
a = np.arange(12).reshape(2,3) =
[[[ 0  1  2]
  [ 3  4  5]]

 [[ 6  7  8]
  [ 9 10 11]]]
```

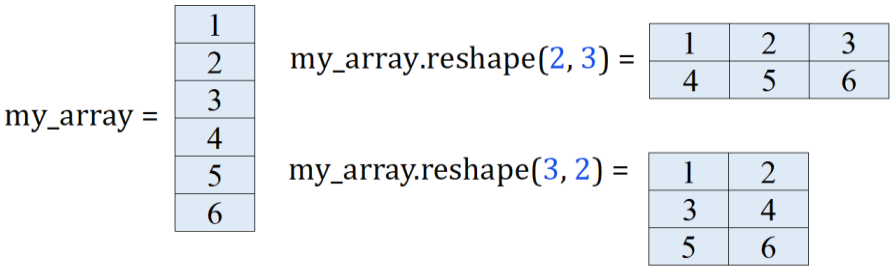


Рисунок 2.6 – Ілюстрація застосування функції numpy.reshape(dimensions).

2.1.6. Операції (оператори) над матрицями різних розмірностей в numpy

Арифметичні операції (оператори) над матрицями різних розмірностей можливі у разі, якщо розмірність однієї з матриць дорівнює одиниці. Це означає, що у матриці лише один стовпчик, чи один рядок. У такому разі для виконання операції numpy використовуватиме правила трансляції (Рисунок 2.7):

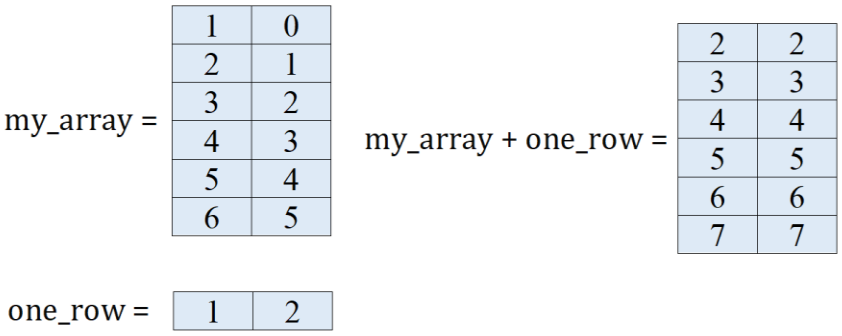


Рисунок 2.7 – Ілюстрація операцій над матрицями різних розмірностей в numpy.

Numpy може зробити всі перераховані вище операції над матрицями однакових розмірностей для будь-яких значень розмірностей (Рисунок 2.8).

my_array = np.array([[[0, 1, 2], [3, 4, 5]], [[-1, -2, -3], [-4, -5, -6]]])

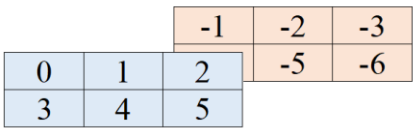


Рисунок 2.8 – Ілюстрація трьохмірних масивів numpy.

Приклад коду зі створення 3-мірного масиву numpy:

```
import numpy as np
a = np.array([[[1, 2, 3], [4, 5, 6], [9, 2, 3]], [[1, 2, 3], [9,
2, 3], [8, 2, 3]]])
print(f'{a = } \n{a.shape = } {a.ndim = }\n')
a = np.zeros((2, 2, 4))
print(f'a = np.zeros((2, 2, 4)) = \n {a}\n')
a = np.ones((3, 2, 3))
print(f'a = np.ones((3, 2, 3)) = \n {a}\n')
a = np.empty((2, 2, 3))
print(f'a = np.empty((2, 2, 3)) = \n {a}\n')
   Результат роботи коду:
a = array([[[1, 2, 3],
        [4, 5, 6],
        [9, 2, 3]],
       [[1, 2, 3],
        [9, 2, 3],
        [8, 2, 3]]])
a.shape = (2, 3, 3) a.ndim = 3

a = np.zeros((2, 2, 4)) =
 [[[0. 0. 0. 0.]
  [0. 0. 0. 0.]]
 [[0. 0. 0. 0.]
  [0. 0. 0. 0.]]]

a = np.ones((3, 2, 3)) =
 [[[1. 1. 1.]
  [1. 1. 1.]]
 [[1. 1. 1.]
  [1. 1. 1.]]
 [[1. 1. 1.]
  [1. 1. 1.]]]

a = np.empty((2, 2, 3)) =
 [[[1. 0. 2.]
  [0. 3. 0.]]
 [[4. 0. 5.]
  [0. 6. 0.]]]
```

У більшості випадків для створення нової розмірності масиву потрібно просто додати кому до параметрів функції numpy (Рисунок 2.9):

```
my_array = np.ones((2, 6, 3))
```

```
my_array0 = np.zeros((2, 6, 3))
```

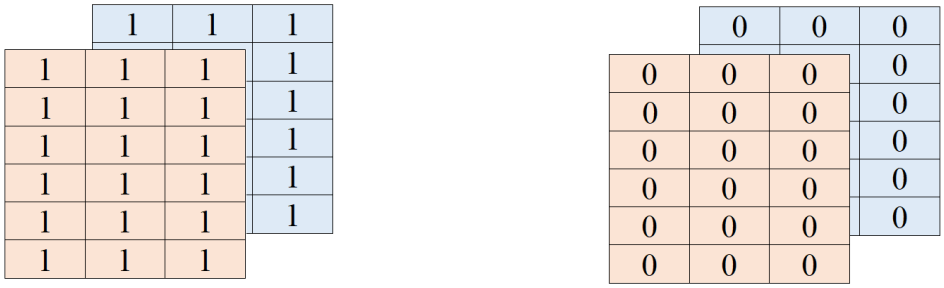


Рисунок 2.9 – Ілюстрація створення трьохмірних масивів в numpy.

Завдання та запитання до підрозділу «Пакет numpy мови програмування python»

1. Створіть масив цілих чисел [[64392, 31655], [32579, 0], [49248, 462], [0, 0]] (4 рядка, 2 стовпчики) і роздрукуйте його атрибути.
2. Відсортуйте масив numpy sampleArray = numpy.array([[34,43,73],[82,22,12],[53,94,66]]). Роздрукуйте результат сортування масиву за другим рядком (тобто в порядку збільшення елементів другого рядка), роздрукуйте результат сортування масиву за другим стовпчиком (в порядку збільшення елементів другого стовпчика).
3. Напишіть код, який здійснює друк заданого масиву. Потім друкує масив після видалення стовпчика 2 на осі 1. Після цього відбувається друк масиву після вставлення стовпчика 2 на місце попередньо видаленого стовпчика.
4. З якого індексу починається індексація масивів numpy?
5. Як можна створити масив numpy?
6. Які переваги роботи з масивами numpy в порівнянні зі списками?
7. Чи може бути NaN елементом масиву numpy?

2.2. Базові методи пакету numpy для статистичного аналізу експериментальних даних

При аналізі експериментальних даних, у регресійному аналізі, в машинному навчанні тощо, важливими є такі величини:

- середнє – це середнє значення набору даних (mean);
- медіана – це значення посередині після сортування набору даних (median);
- мода – це значення в ряду даних, яке повторюється найчастіше (mode);
- стандартне відхилення (середньоквадратичне відхилення) (standard deviation);
- дисперсія (variance), тощо.

2.2.1. Середнє, медіана

Середнє – це середнє значення для заданого набору даних, розраховується за формулою:

$$\bar{x} = \frac{1}{n}\sum_{i=1} x_i \qquad (2.1)$$

Середнє значення (синоніми: математичне сподівання, математичне очікування) – це одна з основних числових характеристик кожної випадкової величини, розраховується для заданого набору даних.

Приклад: бали 15 студентів в групі такі: 99, 86, 87, 88, 111, 86, 103, 87 ,94, 78, 77, 85, 86, 80, 78.

Приклад коду розрахунку середнього значення набору даних – метод numpy.mean():

```
import numpy as np
print('numpy.mean() використовується для знаходження середнього
значення набору даних')
score = [99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86]
mean_score = np.around(np.mean(score))
print(f'mean_scor = np.mean(score) = {mean_score}')
median_score = np.median(score)
print(f'median_score = np.median(score) = {median_score}')
print('Метод numpy.mean() пакету numpy з двомірним масивом')
a = np.array([[6, 8, 6, 0],
              [1, 2, 1, 7],
              [8, 1, 8, 4],
              [5, 3, 0, 5],
              [5, 7, 5, 9]])
print('a = \n', a)
print('axis=None: метод np.mean() аналізує весь масив a')
a_mean = np.mean(a, axis=None)
print(f'a_mean = np.mean(a, axis=None):  {a_mean}')
print('axis=0: метод np.mean() аналізує кожен стовпчик масиву a')
a_mean = np.mean(a, axis = 0)
print(f'a_mean = np.mean(a, axis = 0):  {a_mean}')
print('axis=1: метод np.mean() аналізує кожен рядок масиву a')
a_mean = np.mean(a, axis = 1)
print(f'a_mean = np.mean(a, axis = 1): {a_mean}')
```

Результат роботи коду:

```
numpy.mean() використовується для знаходження середнього значення
набору даних
mean_scor = np.mean(score) = 89.0
Метод numpy.mean() пакету numpy з двомірним масивом a
a =
 [[6 8 6 0]
 [1 2 1 7]
 [8 1 8 4]
 [5 3 0 5]
 [5 7 5 9]]
axis=None: метод np.mean() аналізує весь масив a
a_mean = np.mean(a, axis=None):  4.55
axis=0: метод np.mean() аналізує кожен стовпчик масиву a
a_mean = np.mean(a, axis = 0):  [5.  4.2 4.  5. ]
axis=1: метод np.mean() аналізує кожен рядок масиву a
a_mean = np.mean(a, axis = 1):  [5.   2.75 5.25 3.25 6.5 ]
```

Приклад коду розрахунку медіани набору даних – метод numpy.median():

```
import numpy as np
print('numpy.median() поділяє ряд даних на дві рівні частини,
тобто зліва і справа від цього значення знаходиться однакова
кількість членів упорядкованого ряду даних')
score = [99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86]
median_score = np.median(score)
print(f'median_score = np.median(score) = {median_score}')
print('Метод numpy.median() пакету numpy з двомірним масивом a')
```

```
a = np.array([[6, 8, 6, 0],
              [1, 2, 1, 7],
              [8, 1, 8, 4],
              [5, 3, 0, 5],
              [5, 7, 5, 9]])
print('a = \n', a)
print('axis=None: метод np.median() аналізує весь масив a')
a_median = np.median(a, axis=None)
print(f'a_median = np.median(a, axis=None):  {a_median}')
print('axis=0: метод np.median() аналізує кожен стовпчик масиву
a')
a_median = np.median(a, axis=0)
print(f'a_median = np.median(a, axis=0):  {a_median}')
print('axis=1: метод np.median() аналізує кожен рядок масиву a')
a_median = np.median(a, axis=1)
print(f'a_median = np.median(a, axis=1):  {a_median}')
   Результат роботи коду:
numpy.median() поділяє ряд даних на дві рівні частини, тобто зліва
і справа від
цього значення знаходиться однакова кількість членів
упорядкованого ряду даних
median_score = np.median(score) = 87.0
Метод numpy.median() пакету numpy з двомірним масивом a
a =
 [[6 8 6 0]
 [1 2 1 7]
 [8 1 8 4]
 [5 3 0 5]
 [5 7 5 9]]
axis=None: метод np.median() аналізує весь масив a
a_median = np.median(a, axis=None):  5.0
axis=0: метод np.median() аналізує кожен стовпчик масиву a
a_median = np.median(a, axis=0):  [5. 3. 5. 5.]
axis=1: метод np.median() аналізує кожен рядок масиву a
a_median = np.median(a, axis=1):  [6.  1.5 6.  4.  6. ]
```

2.2.2. Стандартне відхилення (standard deviation)

Стандартне відхилення (середньоквадратичне відхилення) – це число, яке описує, наскільки розкидані значення відносно їх середнього значення. Мале стандартне відхилення означає, що більшість значень близькі до середнього значення. Значне стандартне відхилення означає, що значення поширюються на більш широкий діапазон. Стандартне відхилення обчислюється за формулою:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad (2.2)$$

Припустимо, бали 15 студентів в групі такі: score = [99, 86, 87, 88, 100, 86, 100, 87, 94, 78, 77, 85, 86, 80, 78]. Стандартне відхилення для цього набору даних – 7.5. Це означає, що більшість значень знаходяться в діапазоні 7.5 від середнього значення, що становить 87,4.

Для вибірки даних з більш широким діапазоном: score = [32, 100, 138, 28, 59, 77, 97, 44, 36, 100, 99, 50, 42, 40]. Стандартне відхилення – 32.9. Це означає, що більшість значень знаходяться в діапазоні 32.9 від середнього значення, яке становить 77,4.

Як Ви можете бачити, більш високе стандартне відхилення вказує на те, що значення розподілені в більш широкому діапазоні. Метод обчислення стандартного відхилення пакету numpy std(), назва методу std від англ. standard deviation.

Приклад коду розрахунку стандартного відхилення набору даних – метод numpy.std():

```
import numpy as np
print('numpy.std() розраховує стандартне відхилення
(середньоквадратичне відхилення ) - \n'
      'число, що характеризує розкид значень відносно їх
середнього значення в наборі даних')
# Спочатку використаємо метод mean() numpy, щоб знайти середній
бал
score = [99, 86, 87, 88, 100, 86, 100, 87, 94, 78, 77, 85, 86, 80,
78]
print('score = ', score)
score_mean = np.around(np.mean(score), 1)
print(f'score_mean = np.mean(score): {score_mean}')
score_std = np.around(np.std(score), 1)
print(f'score_std = np.std(score): {score_std}')
score = [32, 100, 138, 28, 59, 77, 97, 44, 36, 100, 99, 50, 42,
40]
print('score = ', score)
score_mean = np.around(np.mean(score), 1)
print(f'score_mean = np.mean(score): {score_mean}')
score_std = np.around(np.std(score), 1)
print(f'score_std = np.std(score): {score_std}')
print('Метод numpy.std() пакету numpy з двомірним масивом a')
a = np.array([[6, 8, 6, 0],
              [1, 2, 1, 7],
              [8, 1, 8, 4],
              [5, 3, 0, 5],
              [5, 7, 5, 9]])
print('a = \n', a)
print('axis=None: метод np.std() аналізує весь масив a')
a_std = np.std(a, axis=None)
print(f'a_std = np.std(a, axis=None):  {a_std}')
print('axis=0: метод np.std() аналізує кожен стовпчик масиву a')
a_std = np.std(a, axis=0)
print(f'a_std = np.std(a, axis=0):  {a_std[0:4]}')
print('axis=1: метод np.std() аналізує кожен рядок масиву a')
a_std = np.std(a, axis=1)
print(f'a_std = np.std(a, axis=1):  {a_std[0:5]}')
```

Результат роботи коду:

```
numpy.std() розраховує стандартне відхилення (середньоквадратичне
відхилення ) - число, що характеризує розкид значень відносно їх
середнього значення в наборі даних
score =  [99, 86, 87, 88, 100, 86, 100, 87, 94, 78, 77, 85, 86,
80, 78]
score_mean = np.mean(score): 87.4
score_std = np.std(score): 7.5
score =  [32, 100, 138, 28, 59, 77, 97, 44, 36, 100, 99, 50, 42,
40]
score_mean = np.mean(score): 67.3
```

```
score_std = np.std(score): 32.9
Метод numpy.std() пакету numpy з двомірним масивом a
a =
 [[6 8 6 0]
 [1 2 1 7]
 [8 1 8 4]
 [5 3 0 5]
 [5 7 5 9]]
axis=None: метод np.std() аналізує весь масив a
a_std = np.std(a, axis=None):  2.8368115905008566
axis=0: метод np.std() аналізує кожен стовпчик масиву a
a_std = np.std(a, axis=0):  [2.28035085 2.78567766 3.03315018
3.03315018]
axis=1: метод np.std() аналізує кожен рядок масиву a
a_std = np.std(a, axis=1):  [3.         2.48746859 2.94745653
2.04633819 1.6583124 ]
```

Варто зазначити, що функція numpy.std() відповідає формулі для розрахунку стандартного відхилення генеральної сукупності. Якщо потрібно оцінити стандартне відхилення вибірки, то слід скористатися пакетом scipy чи використати формулу:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad (2.3)$$

Приклад коду розрахунку стандартного відхилення для набору даних як генеральної сукупності та як вибірки:

```
import numpy as np
import scipy.stats as sts
# Створення вхідних даних
x = np.array([99, 86, 87, 88, 100, 86, 100, 87, 94, 78, 77, 85,
86, 80, 78])
# Визначення середнього значення
mean = np.mean(x)
# Розрахунок суми квадратів (sum of squares)
ss = 0
for score in x:
    ss += (mean - score)**2
std_p = np.sqrt(ss / len(x))
print("Стандартне відхилення для генеральної сукупності,
розраховане за формулою:", std_p)
print("Стандартне відхилення, розраховане за допомогою функції
np.std():", np.std(x))
std_s = np.sqrt(ss / (len(x) - 1))
print("Стандартне відхилення для вибірки, розраховане за
формулою:", std_s)
print("Стандартне відхилення, розраховане за допомогою функції
scipy.stats.tstd():", sts.tstd(x))
    Результат роботи коду:
Стандартне відхилення для генеральної сукупності, розраховане за
формулою: 7.517091636175967
Стандартне відхилення, розраховане за допомогою функції np.std():
7.517091636175967
```

```
Стандартне відхилення для вибірки, розраховане за формулою:
7.780929066818252
Стандартне відхилення, розраховане за допомогою функції
scipy.stats.tstd(): 7.780929066818252
```

2.2.3. Дисперсія (variance)

Дисперсія (variance) є мірою відхилення значень випадкової величини від центру розподілу (середнього). Більші значення дисперсії свідчать про більші відхилення значень випадкової величини від центру розподілу. Дисперсія підраховується як середнє квадратів різниці значень розподілу і середнього розподілу. Тобто, спочатку виявляємо середнє розподілу, потім від кожного значення віднімаємо значення середнього, підносимо його у другу ступінь, для всіх цих квадратів вираховуємо середнє. Це й буде дисперсія. Для того, щоб отримати стандартне відхилення, що вимірюється в тих же одиницях, що й значення змінної, яку ми спостерігаємо, береться квадратний корінь з дисперсії. Дисперсією випадкової величини називають математичне сподівання квадрата відхилення випадкової величини від її математичного очікування.

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x - \bar{x})^2. \quad (2.4)$$

Для розрахунку дисперсії використовується метод var() пакету numpy, назва методу var від англ. variance.

Міри розкиду – такі як дисперсія і стандартне відхилення дають нам розуміння, наскільки добре, наприклад, середнє представляє весь набір даних.

Приклад коду розрахунку дисперсії набору даних – метод numpy.var():

```
import numpy as np
print('numpy.var() розраховує дисперсію, яка є мірою відхилення
значень \n'
      'випадкової величини від центру розподілу (середнього)')
score = [99, 86, 87, 88, 100, 86, 100, 87, 94, 78, 77, 85, 86, 80,
78]
print('score = ', score)
score_var = np.around(np.var(score), 1)
print(f'score_var = np.var(score): {score_var}')

print('Метод numpy.var() пакету numpy з двомірним масивом a')
a = np.array([[6, 8, 6, 0],
              [1, 2, 1, 7],
              [8, 1, 8, 4],
              [5, 3, 0, 5],
              [5, 7, 5, 9]])
print('a = \n', a)
print('axis=None: метод np.var() аналізує весь масив a')
a_var = np.var(a, axis=None)
print(f'a_var = np.var(a, axis=None):  {a_var}')

print('axis=0: метод np.var() аналізує кожен стовпчик масиву a')
a_var = np.var(a, axis=0)
print(f'a_var = np.var(a, axis=0):  {a_var[0:4]}')

print('axis=1: метод np.var() аналізує кожен рядок масиву a')
a_var = np.var(a, axis=1)
print(f'a_var = np.var(a, axis=1):  {a_var[0:5]}\n')
```

```
   Результат роботи коду:
numpy.var() розраховує дисперсію, яка є мірою відхилення значень
випадкової величини від центру розподілу (середнього)
score =  [99, 86, 87, 88, 100, 86, 100, 87, 94, 78, 77, 85, 86,
80, 78]
score_var = np.var(score): 56.5
Метод numpy.var() пакету numpy з двомірним масивом a
a =
 [[6 8 6 0]
 [1 2 1 7]
 [8 1 8 4]
 [5 3 0 5]
 [5 7 5 9]]
axis=None: метод np.var() аналізує весь масив a
a_var = np.var(a, axis=None):  8.0475
axis=0: метод np.var() аналізує кожен стовпчик масиву a
a_var = np.var(a, axis=0):  [5.2  7.76 9.2  9.2 ]
axis=1: метод np.var() аналізує кожен рядок масиву a
a_var = np.var(a, axis=1):  [9.     6.1875 8.6875 4.1875 2.75  ]
```

Аналогічно до стандартного відхилення, numpy.var() розраховує дисперсію для даних як генеральної сукупності. Якщо потрібно розрахувати цей показник для вибірки, то слід використати або пакет scipy, або розрахувати за формулою:

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x-\bar{x})^2 \qquad (2.5)$$

Приклад коду розрахунку дисперсії набору даних як для генеральної сукупності та як для вибірки:

```
import numpy as np
import scipy.stats as sts
# Створення вхідних даних
x = np.array([99, 86, 87, 88, 100, 86, 100, 87, 94, 78, 77, 85,
86, 80, 78])
# Визначення середнього значення
mean = np.mean(x)
# Розрахунок суми квадратів (sum of squares)
ss = 0
for score in x:
    ss += (mean - score)**2
var_p = ss / len(x)
print("Дисперсія для генеральної сукупності, розрахована за
формулою:", var_p)
print("Дисперсія, розрахована допомогою функції np.var():",
np.var(x))
var_s = ss / (len(x) - 1)
print("Дисперсія для вибірки, розрахована за формулою:", var_s)
print("Дисперсія, розрахована за допомогою функції
scipy.stats.tvar():", sts.tvar(x))

Результат роботи коду:
Дисперсія для генеральної сукупності, розрахована за формулою:
56.50666666666667
```

```
Дисперсія, розрахована допомогою функції np.var():
56.50666666666667
Дисперсія для вибірки, розрахована за формулою: 60.542857142857144
Дисперсія, розрахована за допомогою функції scipy.stats.tvar():
60.542857142857144
```

2.2.4. Процентиль (percentile)

Процентилі використовуються у статистиці, щоб розрахувати число, яке описує вибірку значень, які менше за якийсь відсоток значень.

Наприклад, група студентів із 100 осіб пише тест, що складається зі 100 питань. Прохідний поріг ¾, тобто 75 балів. Допустимо, Іван правильно відповів на 70 питань.

Але завдання можна поставити й інакше: потрібно порівняти результати студентів не з прохідним балом у 75 пунктів, а між собою. Наприклад, для об'єктивної оцінки складності тесту, що досягається угрупуванням результатів. Іван отримав 70 правильних відповідей. Чи багато це чи мало в порівнянні з рештою студентів? І це покаже процентиль.

Відсотки ділять усю вибірку на певні частини. Наприклад, п'ятий процентиль охоплює 5% обсягу вибірки (Рисунок 2.10). Припустимо, показник Івана дорівнює п'ятому відсоткові. Це означає, що Іван написав тест краще, ніж 5% студентів, тобто 5 осіб із 100 отримали від нуля до 70 балів, всі інші студенти отримали більше 70 балів. Значить, тест був дуже легкий і поріг у 70 балів можна підвищити. Але в тому ж тесті може бути і зворотна ситуація: результат Івана дорівнює 95-му процентилю (Рисунок 2.10). Це означає, що Іван написав тест краще ніж 95% студентів. Або інакше: лише 5% (5 осіб) набрали понад 70 правильних відповідей, всі інші студенти отримали менше 70 балів. Отже, тест був дуже важким.

Перевага методу ще й у тому, що розбивкою на процентилі можна порівнювати тести з різним числом учасників. Для розрахунку процентилів використовується метод percentile() пакету numpy.

Приклад коду розрахунку процентилів набору даних – метод numpy. percentile() (Рисунок 2.10):

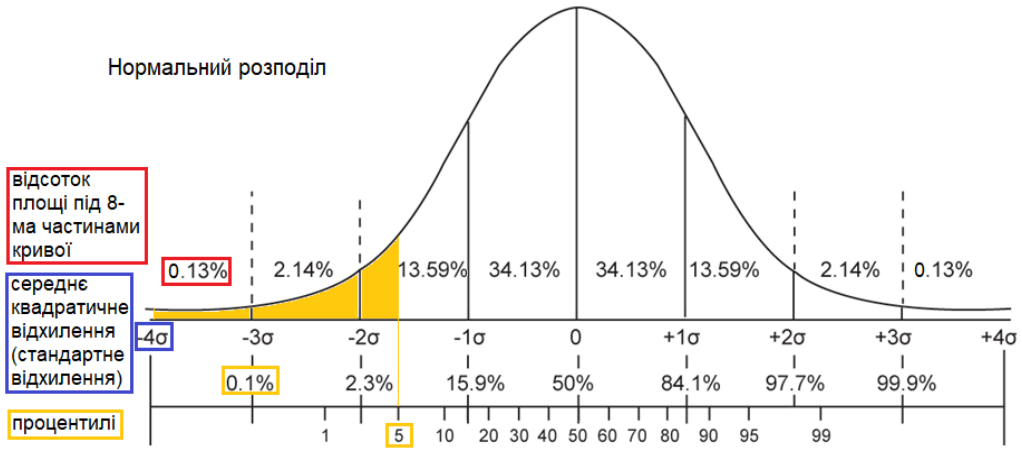
```
import numpy as np
print('numpy.percentile() розраховує число, яке описує вибірку
значень,\n'
      ' які менше за якийсь відсоток значень')
score = [99, 86, 87, 88, 100, 86, 100, 87, 94, 78, 77, 85, 86, 80,
78]
print('score = ', score)
score_perc = np.percentile(score, 75)
print(f'score_perc = np.percentile(score, 75): {score_perc}')
score = [5, 31, 43, 48, 50, 41, 7, 11,15, 39, 80, 82, 32, 2, 8, 6,
25, 36, 27, 61, 31]
print('score = ', score)
score_perc = np.percentile(score, 75)
print(f'score_perc = np.percentile(score, 75): {score_perc}')
print('Метод numpy.percentile() пакету numpy з двомірним масивом
a')
a = np.array([[6, 8, 6, 0],
              [1, 2, 1, 7],
              [8, 1, 8, 4],
              [5, 3, 0, 5],
```

```
                [5, 7, 5, 9]])
print('a = \n', a)
print('axis=None: метод np.percentile() аналізує весь масив a')
a_percentile = np.percentile(a, 75, axis=None)
print(f'a_percentile = np.percentile(a, axis=None):
{a_percentile}')

print('axis=0: метод np.percentile() аналізує кожен стовпчик
масиву a')
a_percentile = np.percentile(a, 75, axis=0)
print(f'a_percentile = np.percentile(a, axis=0):
{a_percentile[0:4]}')

print('axis=1: метод np.percentile() аналізує кожен рядок масиву
a')
a_percentile = np.percentile(a, 75, axis=1)
print(f'a_percentile = np.percentile(a, axis=1):
{a_percentile[0:5]}')
```

а)



б)

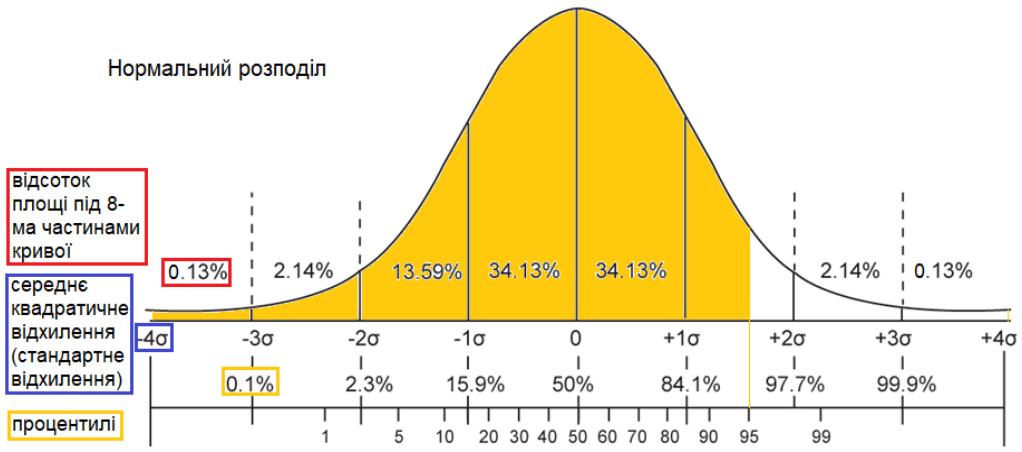


Рисунок 2.10 – а) - ілюстрація поняття 5-ий процентиль: б) - ілюстрація поняття 95-ий процентиль.

```
Результат роботи коду:
numpy.percentile() розраховує число, яке описує вибірку значень,
 які менше за якийсь відсоток значень
score =  [99, 86, 87, 88, 100, 86, 100, 87, 94, 78, 77, 85, 86,
80, 78]
score_perc = np.percentile(score, 75): 91.0
score =  [5, 31, 43, 48, 50, 41, 7, 11, 15, 39, 80, 82, 32, 2, 8,
6, 25, 36, 27, 61, 31]
score_perc = np.percentile(score, 75): 43.0

Метод numpy.percentile() пакету numpy з двомірним масивом a
a =
 [[6 8 6 0]
 [1 2 1 7]
 [8 1 8 4]
 [5 3 0 5]
 [5 7 5 9]]
axis=None: метод np.percentile() аналізує весь масив a
a_percentile = np.percentile(a, axis=None):  7.0
axis=0: метод np.percentile() аналізує кожен стовпчик масиву a
a_percentile = np.percentile(a, axis=0):  [6. 7. 6. 7.]
axis=1: метод np.percentile() аналізує кожен рядок масиву a
a_percentile = np.percentile(a, axis=1):  [6.5  3.25 8.   5.   7.5
]
```

2.2.5. Що таке Q рейтинг журналів?

Рейтинг журналів на основі квартилю (Q1-Q4) розраховується для кожного журналу в кожній його предметній категорії, згідно з імпакт-фактором журналу. Квартиль (одна четверта) означає, що всі журнали у певній категорії поділяються на чотири групи за їхнім рівнем імпакт-фактору. До першого квартиля включають журнали із найбільшими імпакт-факторам у категорії, до останнього четвертого квартилю – із найнижчим.

Рейтинг журналів на основі квартилю виник через те, що імпакт-фактори журналів у різних категоріях дуже різняться, відповідно групування за імпакт-фактором у межах категорії допомагає порівняти журнали із різних категорій. Наприклад, журнал із фізики із імпакт-фактором 76 і журнал із соціології із імпакт-фактором 4 у разі їх порівнювання тільки за імпакт-факторами не порівнювані. Однак, з огляду на те, що цитованість статей із фізики набагато більша, ніж у соціології у рейтингу за квартилями вони можуть потрапити до одного квартилю (але у своїй категорії: фізиці і соціології, відповідно).

Квартиль тісно пов'язаний із процентилем: Q1 – 76-99; Q2 – 51-75; Q3 – 26-50; Q4 ≤ 25.

Для прикладу процентиль журналу у 85 означає, що імпакт-фактор цього журналу більший за імпакт-фактор 85% журналів у цій категорії. Для журналів це не постійний статус. Щорічно проводиться перерахунок результатів цитування, і в сукупності з іншими показниками приймається рішення про залишення або переміщенні журналу в іншу категорію.

2.2.6. Завдання та запитання до теми «Базові методи пакету numpy для статистичного налізу експериментальних даних»

1. Який метод numpy використовується для розрахунку середнього значення?
2. Який метод numpy використовується для розрахунку дисперсії?
3. Напишіть код, який розраховує випадкову похибку вимірювань.

4. Напишіть код, в якому Ви задаєте два масиви: список назв десяти фахових журналів з Вашої спеціальності strJournal = numpy.empty(10, dtype='string') та масив rankingJournal. В першому стовпчику масиву rankingJournal задайте квартиль кожного журналу, в другому стовпчику задайте імпакт- фактор журналу. Розрахуйте та виведіть на екран 75-ий процентиль на основі інформації про імпакт-фактор журналу. Виведіть на екран назви журналів, які мають імпакт фактор менший 75% для даної вибірки журналів. Розрахуйте та виведіть на екран середнє значення квартилю журналу.

2.3. Пакет matplotlib для побудови графіків та діаграм

Matplotlib – бібліотека на мові програмування python для візуалізації даних двовимірною 2D графікою (підтримується також 3D графіка) [23, 26, 28–30].

Matplotlib написана Джоном Хантером (1968–2012), американським нейробіологом, автором книги "Matplotlib: A 2D graphics environment." Computing in science and engineering 9.3 (2007). Бібліотека поширюється на умовах BSD – ліцензії (Berkeley Software Distribution license – Ліцензія на розповсюдження програмного забезпечення Berkeley) [30].

Зображення, які генеруються в різних форматах, можуть бути використані в інтерактивній графіці, для ілюстрацій в публікаціях, графічному інтерфейсі користувача, вебдодатках, для побудови діаграми (plotting).

В документації на matplotlib автор зізнавався, що пакет Matplotlib починався з імітування графічних команд MATLAB, але є незалежним від нього проєктом.

Бібліотека Matplotlib побудована на принципах ООП, але має процедурний інтерфейс pylab, який надає аналоги команд MATLAB [26, 30].

Matplotlib є гнучким, легко конфігурованим пакетом, який разом з NumPy, SciPy надає можливості, подібні до MATLAB. Пакет підтримує багато видів графіків і діаграм [26, 29]:

- Графіки (line plot)
- Діаграми розсіювання (scatter plot)
- Стовпчасті діаграми (bar chart) і гістограми (histogram)
- Секторні діаграми (pie chart)
- Діаграми «стебло–листя» (stem plot)
- Контурні графіки (contour plot)
- Поля градієнтів (quiver)
- Спектральні діаграми (spectrogram)

Користувач може вказати осі координат, сітку, додати підписи і пояснення, використовувати логарифмічну шкалу або полярні координати. Нескладні тривимірні графіки можна будувати за допомогою набору інструментів (toolkit) mplot3d. Існують і інші набори інструментів: для картографії, для роботи з Excel тощо. За допомогою Matplotlib можна створювати і анімовані зображення [26].

Набір підтримуваних форматів зображень, векторних і растрових, можна отримати з словника FigureCanvasBase.filetypes.

Типові підтримувані формати:

- EPS (Encapsulated Post Script) – графічний файл, який описано мовою PostScript. Використовується переважно для друку. Містить як векторну інформацію так і растрову.
- EMF (Enhanced Metafile) – використовується в операційній системі Windows для зберігання буферизованої інформації, що запускається в загальному пулі черги до друку.
- JPEG, PDF
- PNG (Portable Network Graphics) – портативна мережева графіка, растровий формат збереження графічної інформації, що використовує стиснення без втрат. PNG не потребує ліцензії для використання.

- SVG (Scalable Vector Graphics) – масштабована векторна графіка, специфікація мови розмітки, що базується на XML, та формат файлів для двомірної векторної графіки, як статичної, так і анімованої та інтерактивної.

Крім того, на основі класів пакету можна створювати й інші модулі. Наприклад, для генерування спарклайнів. Спарклайн (spark line – іскрова лінія) – це дуже маленька лінійна діаграма, зазвичай намальована без осей чи координат.

Matplotlib – це пакет для побудови графіків поліграфічної якості [26]. Для роботи з пакетом Matplotlib необхідно його завантажити:

```
import matplotlib                          # завантаження пакету matplotlib
print(matplotlib.__version__)  # друк версії пакету matplotlib
# завантаження модуля font із бібліотеки tkinter для підтримки шрифтів підписів на
# графіках
import tkinter.font as font
# завантаження пакету numpy для роботи з масивами при побудові графіків
import numpy as np
# завантаження основного модуля pyplot пакету matplotlib роботи з графіками та
# діаграмами
import matplotlib.pyplot as plt
```

2.3.1. Властивості растрових та векторних зображень

Властивості растрових зображень:

- Розмір – ширина та висота. Вимірюється в пікселях, сантиметрах, дюймах.
- Роздільність – кількість пікселів на одиницю довжини. Вимірюється в пікселях на дюйм (dpi) або пікселях на см. Чим більше роздільність – тим чіткіше зображення, але і більший розмір файлу.
- Глибина кольору – кількість бітів, що використовується для кодування кольору одного пікселя. Вимірюється в бітах на піксель (bpp); 8 біт – 256 кольорів, 16 біт – 65536 кольорів, 24 біт – більше 16 млн., 32 біт – більше 4 млрд.
- Основні властивості векторних зображень:
- Види та кількість графічних примітивів, тобто окремих геометричних фігур: відрізків, багатокутників, кривих, овалів, з яких будується зображення.
- Кількість кольорів, що використовуються.

На відміну від растрових, векторні зображення складаються вже не з пікселів, а з безлічі опорних точок і кривих, які їх з'єднують. Векторне зображення описується математичними формулами і, відповідно, не вимагає наявності інформації про кожний піксель. Скільки не збільшуй масштаб векторного зображення, Ви ніколи не побачите пікселів.

Найпопулярніші векторні формати: SVG, AI. AI підтримує практично всі програми, пов'язані з векторною графікою, є найкращим посередником при передачі зображень з однієї програми в іншу, відрізняється найбільшою стабільністю і сумісністю з мовою PostScript, на яку орієнтуються практично всі видавничо-поліграфічні додатки.

Векторна графіка використовується для ілюстрацій, іконок, логотипів і технічних креслень, але складна для відтворення фотореалістичних зображень. Найпопулярніший редактор векторної графіки – Adobe Illustrator.

Розглянемо переваги і недоліки растрової графіки.

Переваги:

- Можливість створити зображення будь-якої складності – з величезною кількістю деталей і широкою гамою кольорів.
- Растрові зображення найбільш поширені.
- Працювати з растровою графікою простіше, так як механізми її створення та редагування більш звичні і поширені.

Недоліки:

- Великий обсяг пам'яті, який займають такі зображення: чим більше «розмір» зображення, тим більше в ньому пікселів і, відповідно, тим більше місця потрібно для зберігання / передачі такого зображення.
- Неможливість масштабування: растрове зображення неможливо масштабувати без втрат. При зміні розміру оригінального зображення неминуче (в результаті процесу інтерполяції) відбудеться втрата якості.

Розглянемо також переваги і недоліки векторної графіки.

Переваги:

- Малий обсяг пам'яті, який займають такі зображення – векторні зображення мають менший розмір, так як містять в собі малу кількість інформації.
- Векторні зображення відмінно масштабуються – можна нескінченно змінювати розмір зображення без втрат якості.

Недоліки:

- Щоб відобразити векторне зображення потрібно зробити ряд обчислень, відповідно, складні зображення можуть вимагати підвищених обчислювальних потужностей.
- Не кожне зображення може бути представлене в векторному вигляді: для складного зображення з широкою гамою кольорів може знадобитися безліч точок і кривих, що зведе «нанівець» всі переваги векторної графіки.
- Процес створення і редагування векторної графіки відрізняється від звичної складністю моделі – для роботи з векторною графікою потрібні додаткові знання.

Інтерфейс модуля pylab пакету matplotlib дозволяє легко використовувати matplotlib досвідченими користувачами MATLAB.

Нижче наведені деякі переваги використання matplotlib, як аналогу MATLAB:

- Вбудована підтримка SVG (Scalable Vector Graphics) – масштабована векторна графіка, мова розмітки на основі XML для опису двовимірної векторної графіки, що описує зображення, які можна візуалізувати у будь-якому розмірі та розроблена спеціально для того, щоб добре працювати з іншими вебстандартами, включаючи CSS, DOM, JavaScript та SMIL.
- Це відкрите програмне забезпечення, яке розроблено робочою групою SVG Working Group організації World Wide Web Consortium.

2.3.2. Графіки в matplotlib

Графіки в matplotlib, будуються всередині об'єкта – екземпляру класу Figure. Створити новий екземпляр fig класу Figure:

```
fig = plt.figure()
```

З'являється нове порожнє вікно. Рисунки в matplotlib підтримують схему нумерації аналогічно MATLAB.

Спочатку потрібно створити один або кілька графіків (панелей) за допомогою методу, наприклад, add_subplot() [31]:

```
ax1 = fig.add_subplot (1, 1, 1)
```

Рисунок 2.11 ілюструє додавання декількох панелей на рисунок.

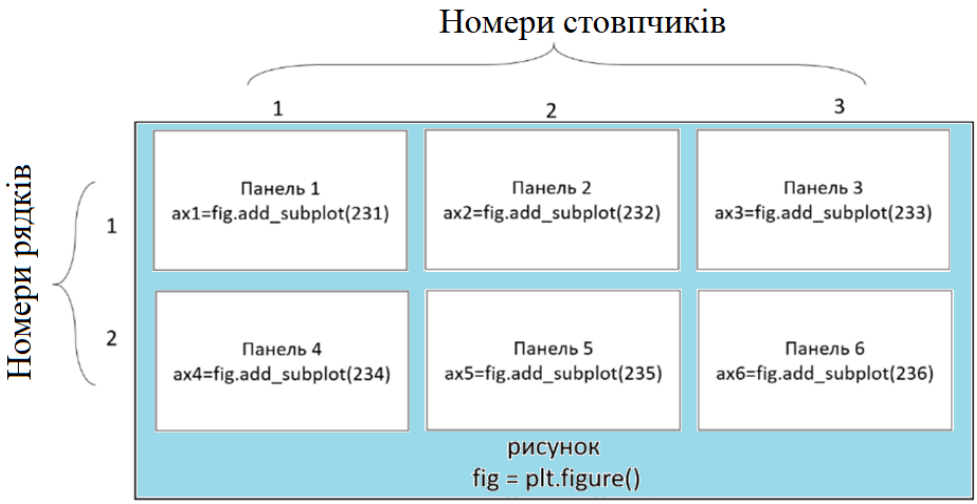


Рисунок 2.11 – Додавання декількох панелей на рисунок, а саме діаграма розташування шести панелей з іменами ax1, …, ax6 в три стовпчики і два рядки на рисунку fig.

Далі ми будемо використовувати таку термінологію (Рисунок 2.11, Рисунок 2.12):

- Рисунок (Figure) – це область самого верхнього рівня, та область на якій розташовуються рисунки. Така область може містити кілька панелей Axes.
- Панель (Axes) – це та область, на якій відображаються графіки, діаграми тощо та допоміжні атрибути (лінії сітки, мітки, показники тощо), яка часто супроводжується викликом методу add_subplot або subplots, які і поміщають панелі (AxesSubplot) на регулярну сітку на рисунку [31].
- Кожна панель (AxesSubplot) містить осі координат (XAxis і Yaxis), які містять мітки та інші допоміжні атрибути.

Спочатку роботи створюємо об’єкт fig класу Figure та виводимо рисунок на екран (Рисунок 2.13):

```
import matplotlib
# pyplot – основний модуль пакету Matplotlib для оформлення даних
# (графіків, діаграм, тощо)
import matplotlib.pyplot as plt
fig = plt.figure()          # створюємо об’єкт fig класу Figure
plt.show()                  # виводимо рисунок на екран
```

У рядку fig = plt.figure() цього прикладу ми створили об’єкт fig класу Figure (екземпляр класу Figure).

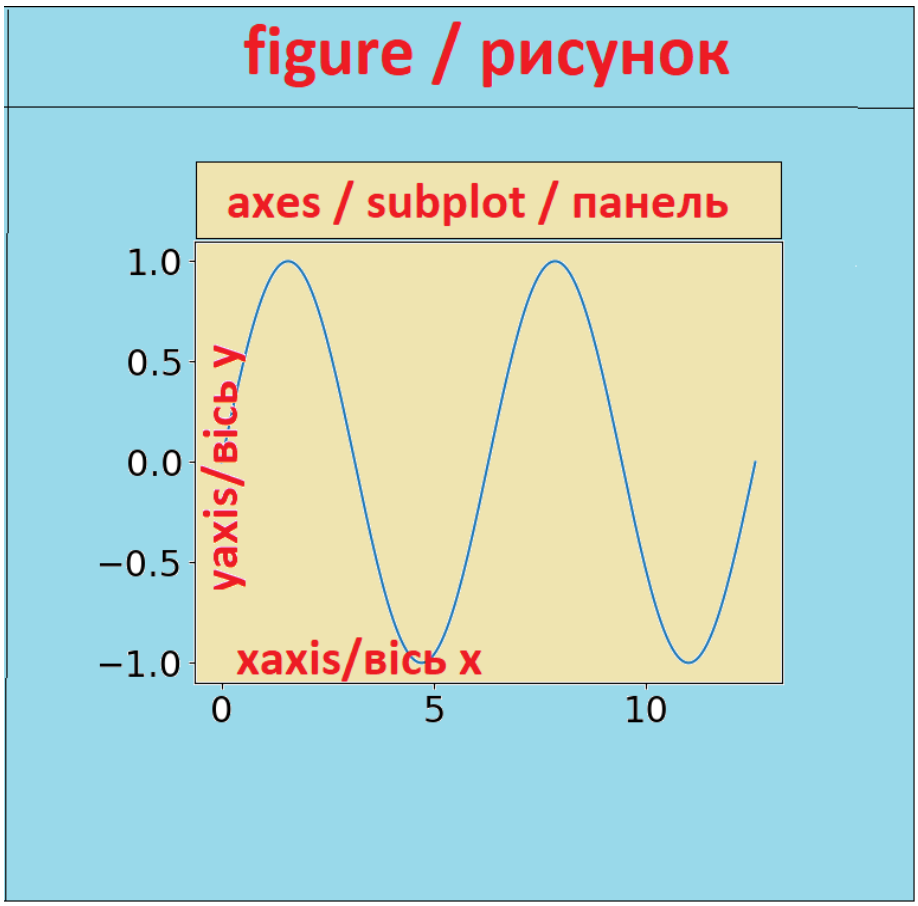


Рисунок 2.12 – Схема розташування областей: рисунок (figure), панель (axes), осі координат (xaxis і yaxis).

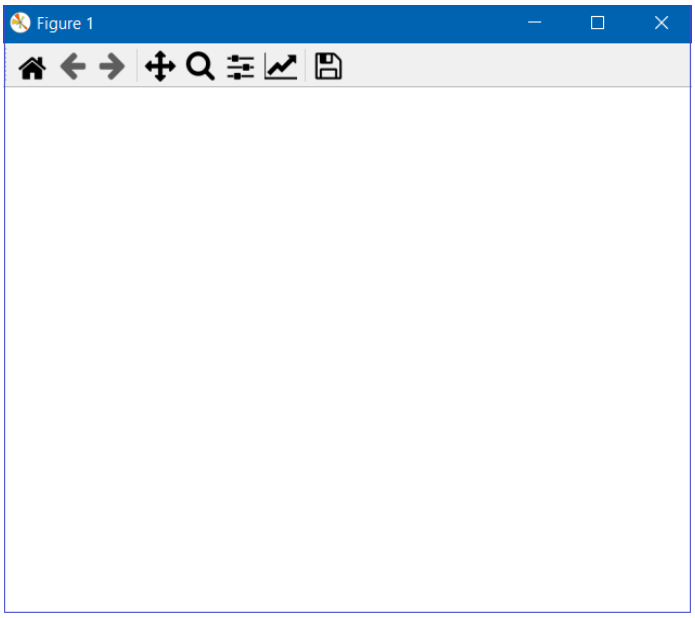


Рисунок 2.13 – Результат роботи коду із попереднього прикладу.

Зробимо те ж саме спочатку, тільки додаємо панель (синоніми Axes і Subplot):

```
import matplotlib.pyplot as plt
fig = plt.figure()              # створюємо рисунок\
ax = fig.add_subplot(111)       # додаємо панель (синоніми Axes і
```

```
Subplot)
print(type(fig), type(ax))
plt.show()                          # виводимо рисунок на екран
```

У рядку ax = fig.add_subplot (111) ми додали до Рисунку (Figure) панель (AxesSubplot), метод add_subplot розміщує панель (AxesSubplot) на рисунку (Figure). За замовчуванням діапазон зміни значень по осям абсцис та ординат від 0 до 1. Параметр, 111 (або 1,1,1) – це один рядок, один стовпчик і перша (єдина) Панель (AxesSubplot) на Рисунку (Figure), всі цілі числа мають бути меншими за 10.

Результат роботи коду представлено на Рисунок 2.14.

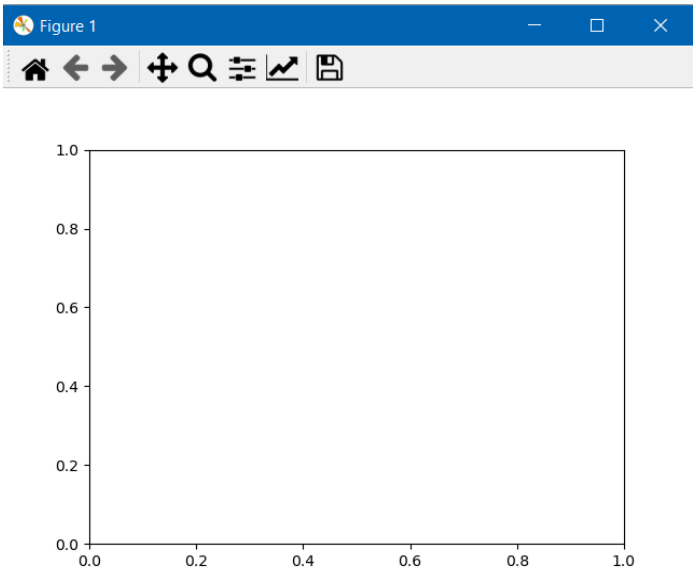


Рисунок 2.14 – Результат роботи коду із попереднього прикладу: <class 'matplotlib.figure.Figure'> <class 'matplotlib.axes._subplots.AxesSubplot'>.

В тому що Рисунок (Figure) і Панель (Axes) – це різні області можна легко переконатися, якщо змінити їх колір (Рисунок 2.15):

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
# використовуємо метод set() і параметр facecolor
fig.set(facecolor = 'green')
ax.set(facecolor = 'red')
plt.show()  # виводимо рисунок на екран
```

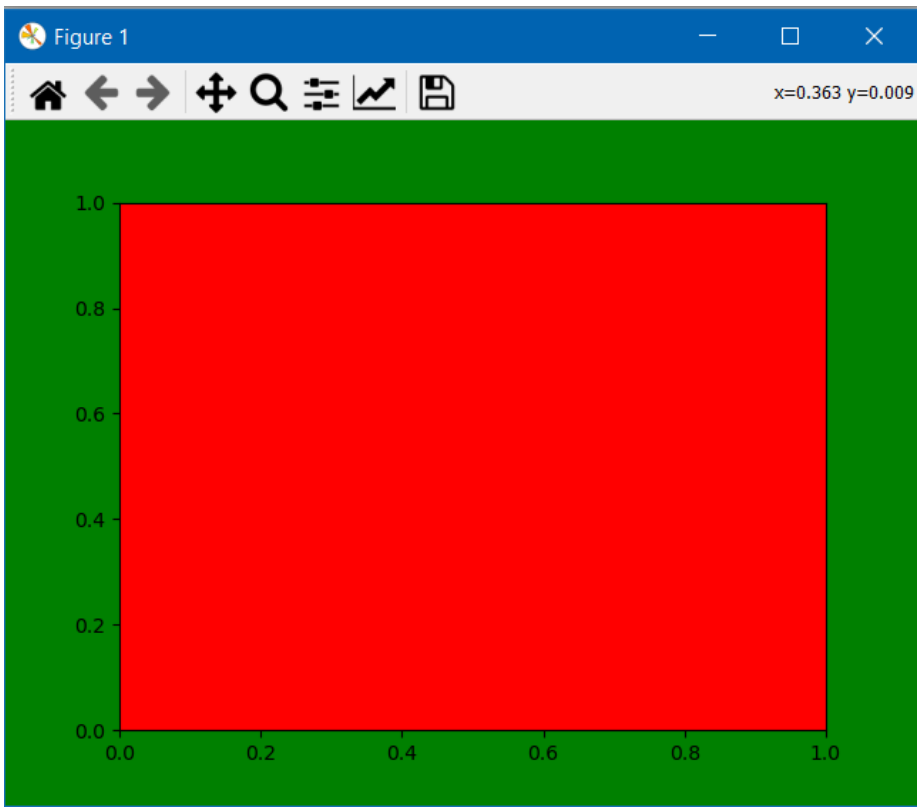


Рисунок 2.15 – Результат роботи коду із попереднього прикладу.

В наступному прикладі розглянемо метод set() та його параметри (Рисунок 2.16):

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
fig.set(facecolor='green')
ax.set(facecolor='red',
       xlim=[-10, 10],
       ylim=[-2, 2],
       title='Основи роботи з matplotlib',
       xlabel='вісь абсцис (XAxis)',
       ylabel='вісь ординат (YAxis)')
plt.show()
```

Розглянемо тепер відображення даних на графіку. Для більшості графіків: ліній, гістограм, кругових діаграм відображення даних відбувається на Панелі (AxesSubplot)). Тому, для відображення даних на Панелі (AxesSubplot)) необхідно використовувати які-небудь з її методів.

Цих методів ціла низка, але спочатку розглянемо два:

- plot – лінія
- scatter – точки

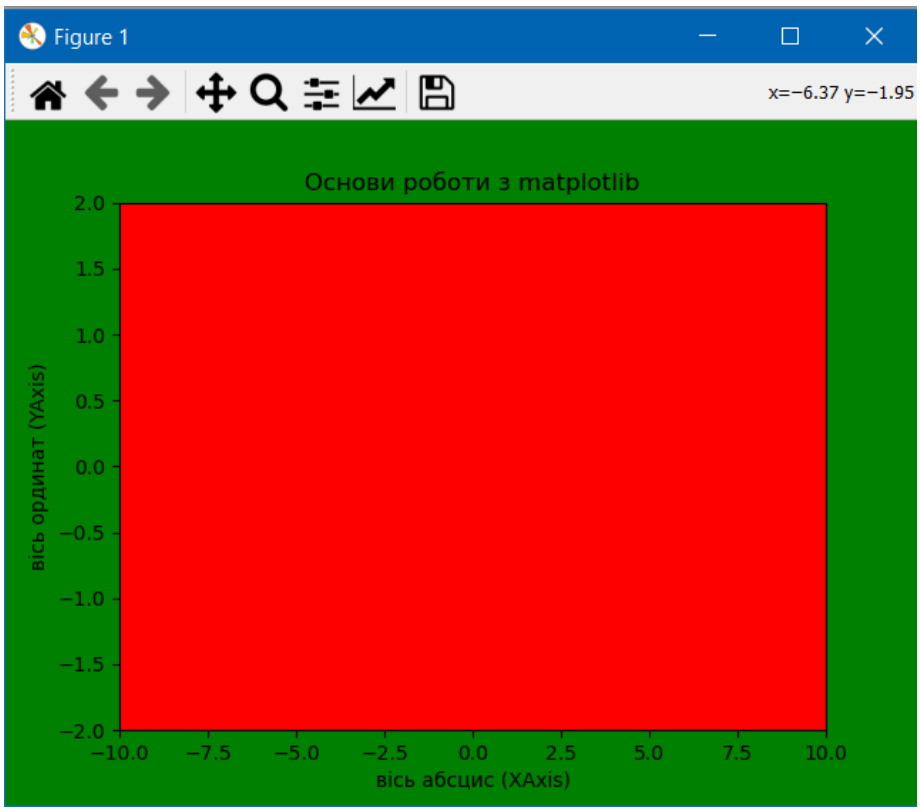


Рисунок 2.16 – Результат роботи коду із попереднього прикладу.

В наступному прикладі будується графік, на якому перший набір даних відображується точками, з’єднаними суцільними лініями, а другий точками, не з’єднаними лініями (Рисунок 3.17):

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot([0, 1, 2, 3, 4], [0, 6, 7, 15, 19])
ax.scatter([0, 1, 2, 3, 4], [1, 3, 8, 12, 27])
plt.show()
```

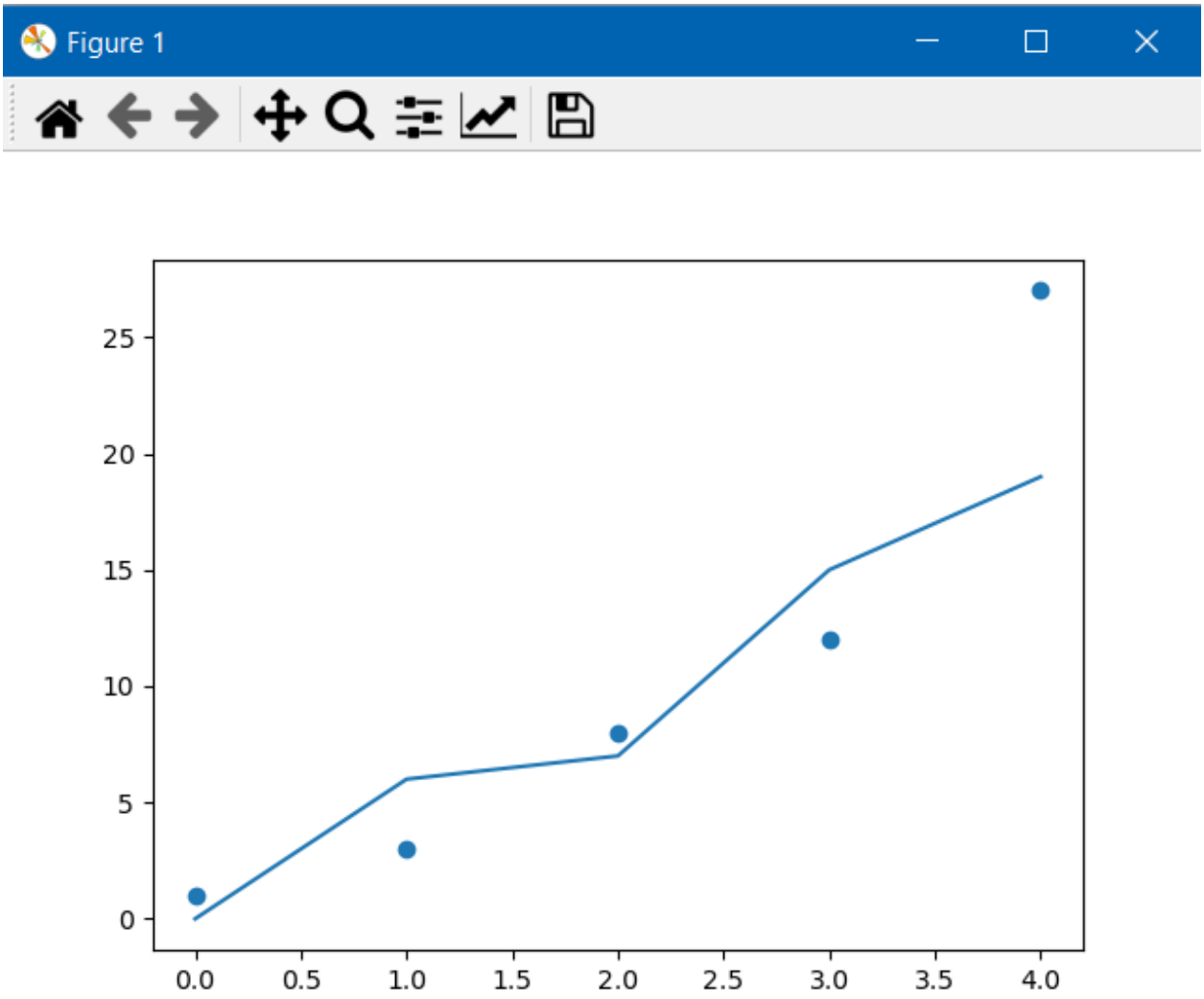


Рисунок 2.17 – Результат роботи коду із попереднього прикладу.

Відображені на графіку дані так само підтримують найрізноманітніші параметри зовнішнього вигляду (Рисунок 2.18):

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot([0, 1, 2, 3, 4], [0, 6, 7, 15, 19],
                color='black', linewidth=5)

ax.scatter([0, 1, 2, 3, 4], [1, 3, 8, 12, 27],
                color='blue',  marker ='*')
plt.show()
```

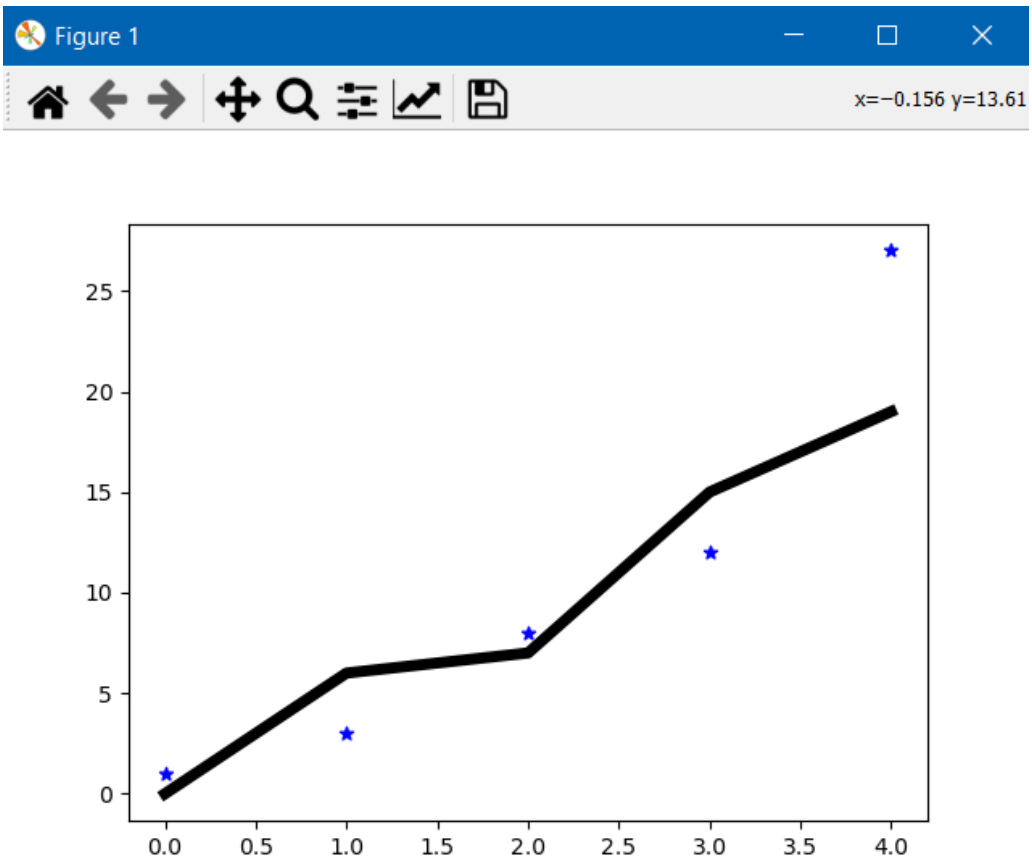


Рисунок 2.18 – Результат роботи коду із попереднього прикладу.

Наступний приклад показує, як можна створити декілька панелей (Axes) на одному рисунку (Figure) (Рисунок 2.19):

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax_1 = fig.add_subplot(2, 2, 1)  # 2 рядка панелей, 2
                                 # стовпчика панелей,
                                 # 1-ша панель
ax_2 = fig.add_subplot(2, 2, 2)  # 2 рядка панелей, 2
                                 # стовпчика панелей,
                                 # 2-га панель

ax_3 = fig.add_subplot(2, 2, 3)  # 2 рядка панелей, 2
                                 # стовпчика панелей,
                                 # 3-тя панель

ax_4 = fig.add_subplot(2, 2, 4)  # 2 рядка панелей, 2
                                 # стовпчика панелей,
                                 # 4-та панель

ax_1.set(title = 'ax_1', xticks=[], yticks=[])
ax_2.set(title = 'ax_2', xticks=[], yticks=[])

ax_3.set(title = 'ax_3', xticks=[], yticks=[])
ax_4.set(title = 'ax_4', xticks=[], yticks=[])
plt.show()     # виводимо рисунок на екран
```

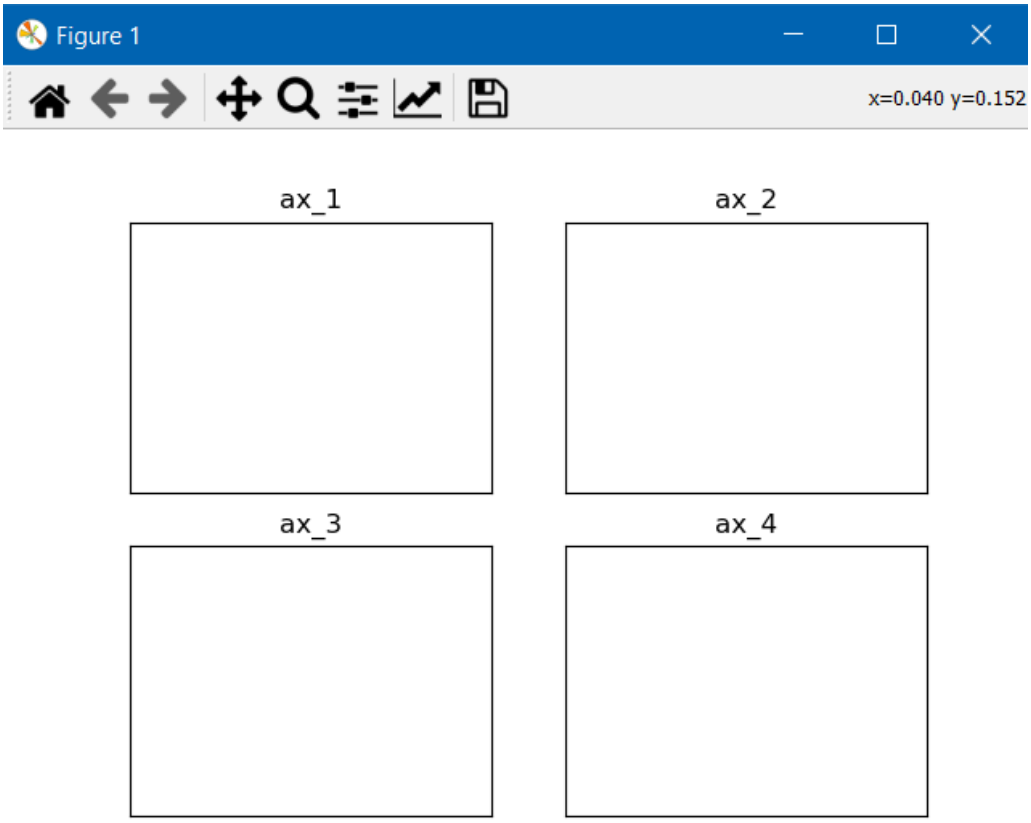


Рисунок 2.19 – Результат роботи коду із попереднього прикладу.

Розглянемо параметри xticks, yticks in matplotlib: xticks, yticks – це список місць розташування позначок на осі x та y, відповідно. Передача порожнього списку видаляє всі позначки (Рисунок 2.20).

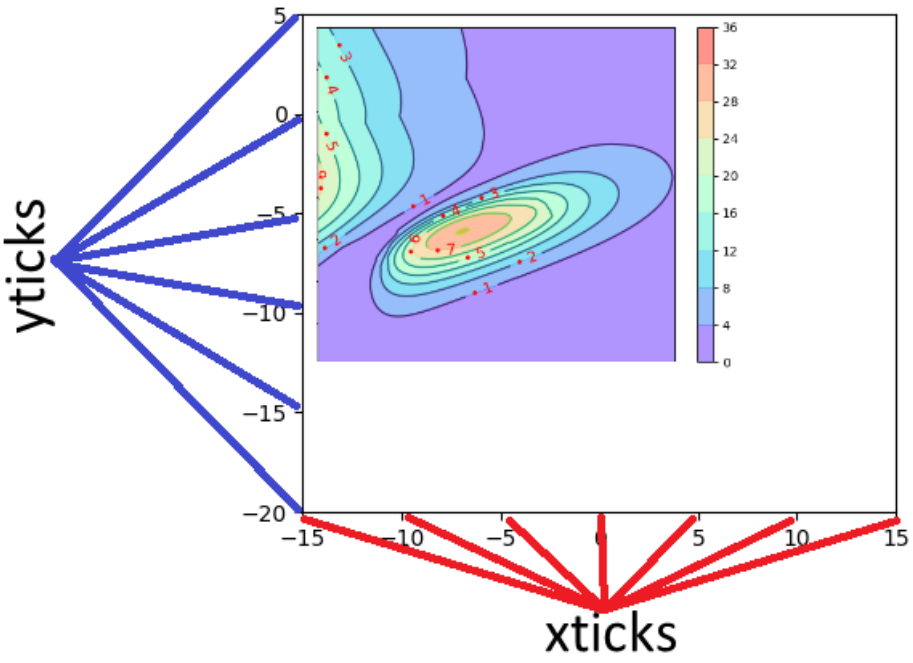


Рисунок 2.20 – Схематичне зображення параметрів xticks, yticks.

Наступний приклад ілюструє, що заповнювати всю область Figure панелями AxesSubplot не обов'язково, також цей приклад показує результат передачі порожніх списків в якості параметрів xticks, yticks (Рисунок 2.21):

```
import matplotlib.pyplot as plt
fig = plt.figure()
# для всіх панелей (Axes)
# 3 рядки і 2 стовпчики
ax_1 = fig.add_subplot(3, 2, 1)
ax_2 = fig.add_subplot(3, 2, 4)
ax_3 = fig.add_subplot(3, 2, 5)
ax_1.set(title = 'ax_1', xticks=[], yticks=[])
ax_2.set(title = 'ax_2', xticks=[], yticks=[])
ax_3.set(title = 'ax_3', xticks=[], yticks=[])
plt.show()
```

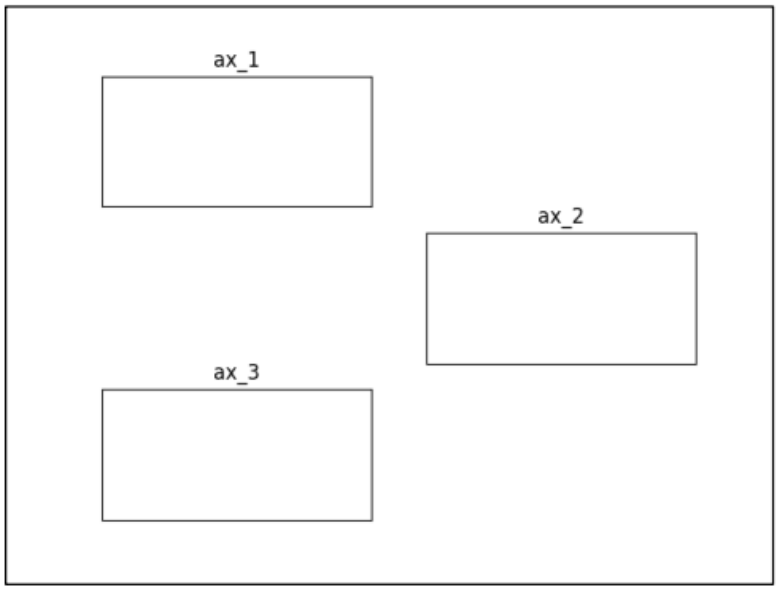


Рисунок 2.21 – Результат роботи коду із попереднього прикладу.

Кожен окремий виклик add_subplot() виконує розбивку Рисунку (Figure), так як зазначено в його параметрах і не залежить від попередніх заданих параметрів. Тому, як показано в наступному прикладі, деякі панелі можуть перекривати одна одну, а деякі можуть не зображуватися (Рисунок 2.22):

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax_1 = fig.add_subplot(3, 1, 1)
ax_2 = fig.add_subplot(6, 3, 3)
ax_3 = fig.add_subplot(3, 3, 4)
ax_4 = fig.add_subplot(3, 3, 6)
ax_5 = fig.add_subplot(3, 4, 10)
ax_6 = fig.add_subplot(5, 5, 25)
ax_1.set(title = 'ax_1', xticks=[], yticks=[])
ax_2.set(title = 'ax_2', xticks=[], yticks=[])
ax_3.set(title = 'ax_3', xticks=[], yticks=[])
ax_4.set(title = 'ax_4', xticks=[], yticks=[])
ax_5.set(title = 'ax_5', xticks=[], yticks=[])
```

```
ax_6.set(title = 'ax_6', xticks=[], yticks=[])
plt.show()
```

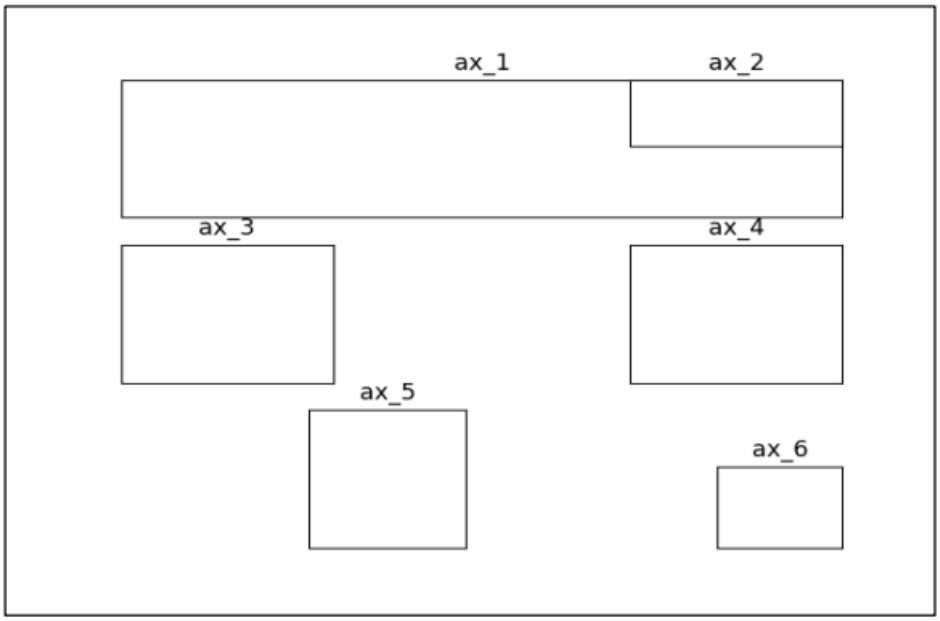


Рисунок 2.22 – Результат роботи коду із попереднього прикладу.

У попередньому прикладі, так само як і раніше, ми спочатку створили Рисунок (Figure), а потім за допомогою команди fig.add_subplot() почали додавати, одну за одною Панелі (Axes) (ax_1, ax_2, ax_3, ax_4).

Причому кожна Панель (Axes) є незалежною від інших, тобто на них можуть бути нанесені найрізноманітніші графіки і встановлені найрізноманітніші параметри зовнішнього вигляду.

Метод add_subplot() розбиває Рисунок (Figure) на вказану кількість рядків і стовпчиків. Після такого розбиття Рисунок (Figure) можна уявити як таблицю (або координатну сітку).

Потім Панель (Axes) поміщається в зазначене місце координатної сітки. Для цього методу add_subplot() необхідно всього три числа, які ми і передаємо йому в якості параметрів:

- перше – кількість рядків;
- друге – кількість стовпчиків,
- третє – індекс Панелі.

Індексування отриманих Панелей починається з 1 (одиниці) з лівого верхнього кута, виконується через порядок зліва направо і закінчується в правому нижньому кутку.

Метод add_subplot(), як було показано вище, дозволяє розташовувати дані (графіки) як вам необхідно. Панелі (Axes) можуть перекривати одна одну, бути різного розміру або розділені деяким простором та розміщуватися в довільних місцях. Звичайно, такий спосіб розміщення деякої кількості панелей (Axes) на рисунку (Figure) досить гнучкий.

Але можна використовувати для таких цілей і метод plt.subplots(nrows, ncols). Але при цьому панелі (Axes) не можуть перекривати одна одну, бути різного розміру або розділені деяким простором та розміщуватися в довільних місцях.

За замовчуванням кількість рядків і стовпчиків в методі subplots дорівнює 1, що зручно для швидкого створення Рисунку (Figure) з однією Панеллю (Axes). Далі, в таких простих випадках, ми будемо дуже часто користуватися саме цим рядком

fig, ax = plt.subplots() – це скорочує код, але не применшує його розуміння. Ми як і раніше створюємо рисунок (Figure) і поміщаємо на ньому панелі (Axes) (Рисунок 2.23):

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()     # один рядок замість двох:
fig = plt.figure()
ax = fig.add_subplot(111)
ax.set(title='Axes')
plt.show()
```

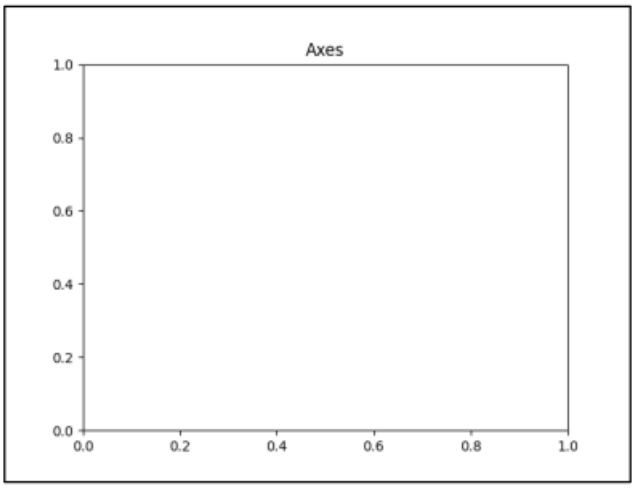


Рисунок 2.23 – Результат роботи коду із попереднього прикладу.

Функція plt.subplots(nrows, ncols) створює кортеж з двох елементів (fig, axes):

- Область рисунку Figure
- Масив об'єктів NumPy, що складається з nrows рядків і ncols стовпчиків.

Кожен елемент цього масиву є окремою панеллю (Axes), до якої можна звернутися по її індексу в даному масиві. Для подальшої роботи з даними панелями необхідно розпакувати даний кортеж, що ми і робимо в рядку:
fig, axes = plt.subplots(nrows = 2, ncols = 2)
Тепер fig – це рисунок (Figure), а axes – це масив NumPy, елементами якого є об'єкти панелі (Axes). Далі, ми встановлюємо для кожної панелі Axes свій заголовок:

```
axes[0, 0].set(title='axes[0, 0]')
axes[0, 1].set(title='axes[0, 1]')
axes[1, 0].set(title='axes[1, 0]')
axes[1, 1].set(title='axes[1, 1]')
```

Можна створювати графіки і без завдання Figure та Axes. Справа в тому, що майже всі методи класу AxesSubplot присутні в модулі pyplot. Наприклад, при виклику plt.title('name') модуль pyplot викликає ax.set_title('name'). Можна сказати, що модуль pyplot створює Рисунок (Figure) і Панель (Axes Subplot) автоматично. Фактично ми можемо переписати весь наш код таким чином (Рисунок 2.24):

```
import matplotlib.pyplot as plt
plt.plot([0, 1, 2, 3, 4], [0, 6, 7, 15, 19], linewidth = 3)
plt.scatter([0, 1, 2, 3, 4], [1, 3, 8, 12, 27],
      color = 'orange')
plt.title('name')
plt.show()
```

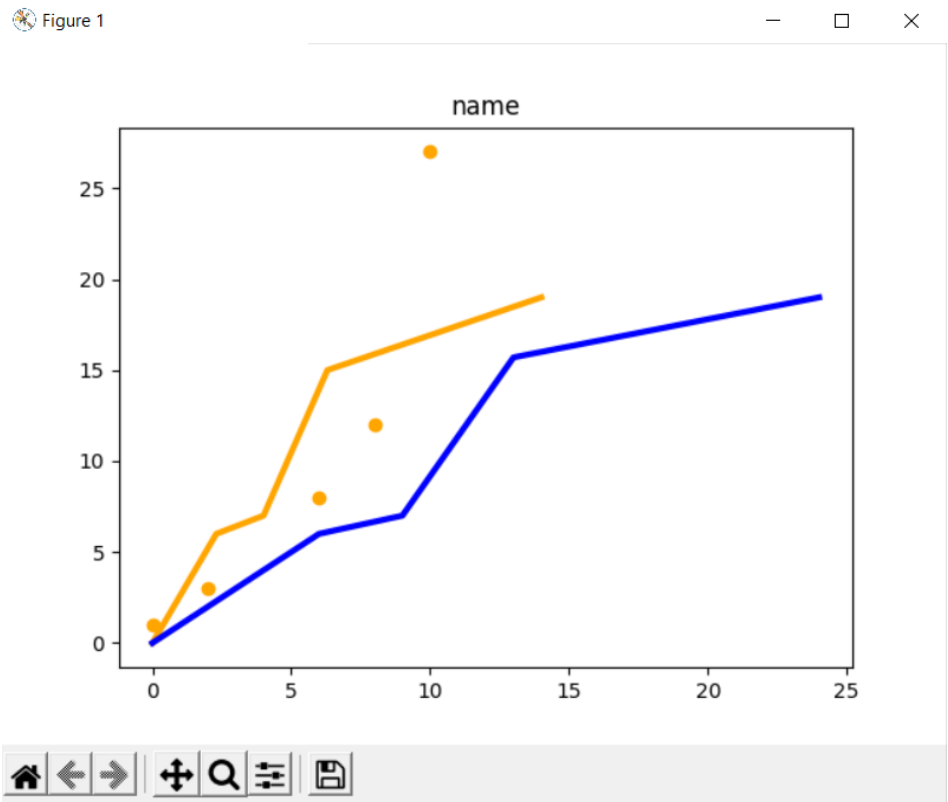


Рисунок 2.24 – Результат роботи коду із попереднього прикладу.

Використання методів класу AxesSubplot и модулю pyplot має такі особливості:

- Якщо потрібно побудувати графіки з невеликими скриптами, щоб щось візуалізувати, то від стислості скриптів ми тільки виграємо.
- Однак, якщо нам доведеться працювати з декількома панелями (AxesSubplot) або доведеться створювати великі скрипти для побудови складної графіки, то використання явних визначень рисунку (Figure) і панелі (AxesSubplot) зробить код більш гнучким, очевидним і зрозумілим, нехай навіть за рахунок збільшення його розміру.

Тепер на низці прикладів проілюструємо налаштування параметрів графіків (Рисунок 2.25):

```
import matplotlib
import numpy as np          # робота з масивами при побудові графіків
import matplotlib.pyplot as plt
# Зображуємо лінію на графіку з точки з координатами від точки(0,0)
# до точки з координатами (6,250)
xpoints = np.array([0, 6])
ypoints = np.array([0, 250])
plt.plot(xpoints, ypoints)
plt.show()
#Зображуємо лінію на графіку з точки з координатами (1, 3)
# до точки з координатами (8, 10)
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
plt.plot(xpoints, ypoints)
plt.show()
```

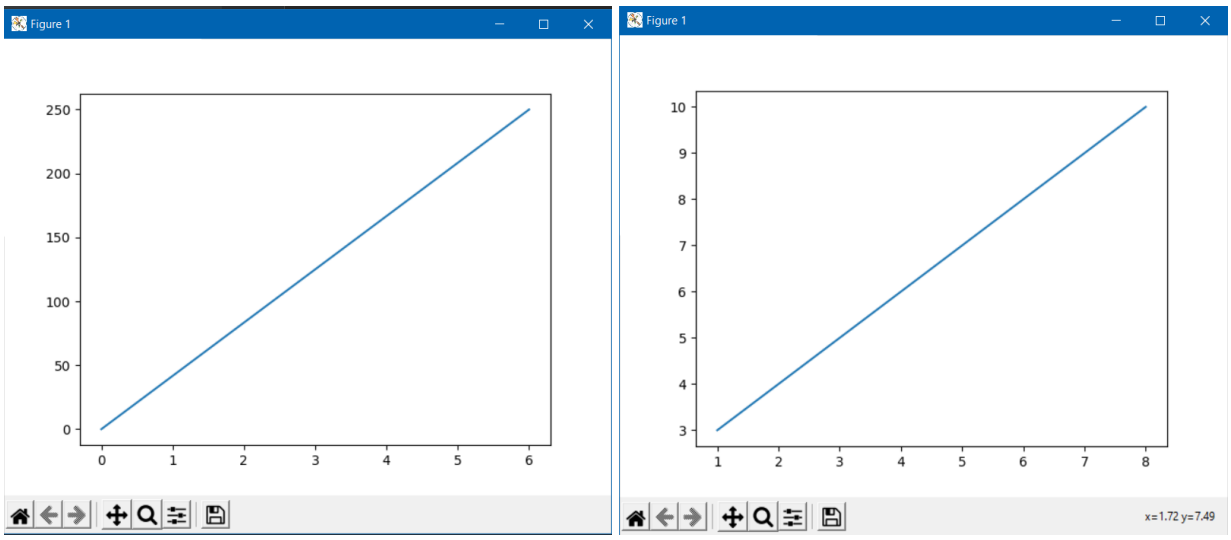


Рисунок 2.25 – Результат роботи коду із попереднього прикладу.

Наступний приклад показує зображення тільки маркерів (Рисунок 2.26):

```
import numpy as np
import matplotlib.pyplot as plt
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
# за замовчуванням відображається лінія
# для того, щоб зобразити тільки маркери, використаємо
# позначення параметру 'o', який означає 'кільце'
plt.plot(xpoints, ypoints, 'o')
plt.show()
```

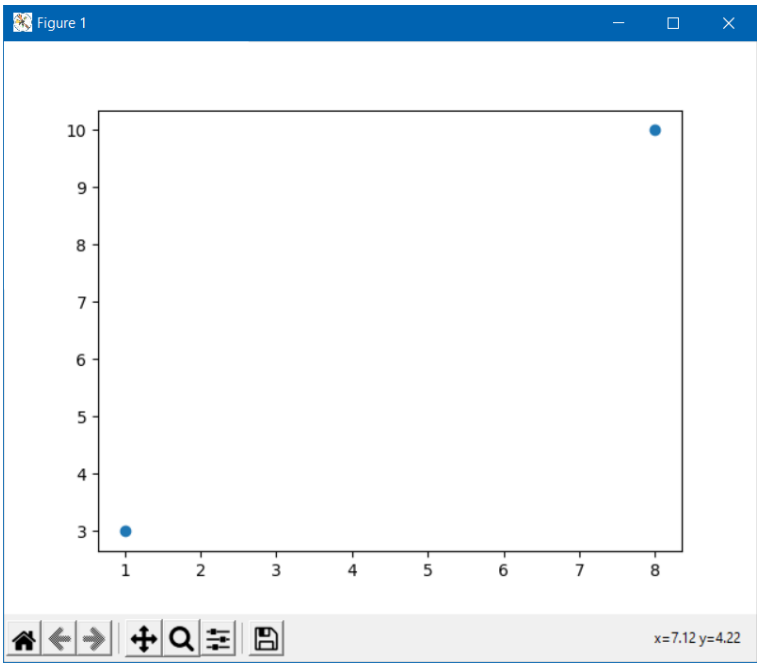


Рисунок 2.26 – Результат роботи коду із попереднього прикладу.

Якщо використаємо ключове слово marker= ‘*’, отримаємо лінію.
Наступний приклад показує зображення похибки на графіку (Рисунок 2.27):

```
import numpy as np
import matplotlib.pyplot as plt
# масиви з координатами (1, 3), (2, 8), (6, 1) і (8, 10)
xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])
xerr = np.array([0.4, 0.5, 0.6, 0.3])
yerr = np.array([0.6, 0.7, 0.5, 0.8])
plt.errorbar(xpoints, ypoints, yerr, xerr, capsize=4)
plt.show()
```

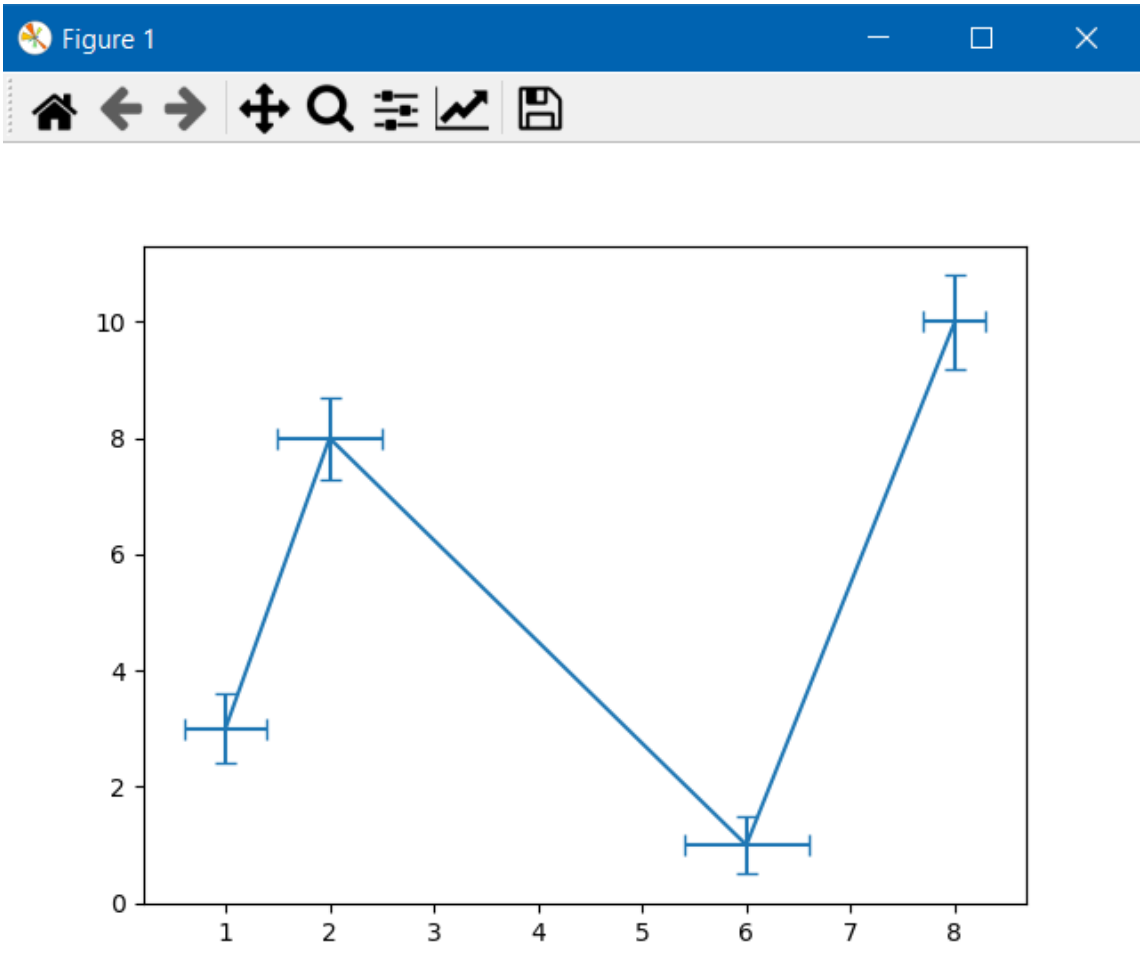


Рисунок 2.27 – Результат роботи коду із попереднього прикладу.

Якщо потрібно зобразити тільки похибку даних вздовж осі ординат, то модифікуємо попередній код (Рисунок 2.28):

```
import numpy as np
import matplotlib.pyplot as plt
# масиви з координатами (1, 3), (2, 8), (6, 1) і (8, 10)
xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])
yerr = np.array([0.6, 0.7, 0.5, 0.8])
plt.errorbar(xpoints, ypoints, yerr, capsize=4)
plt.show()
```

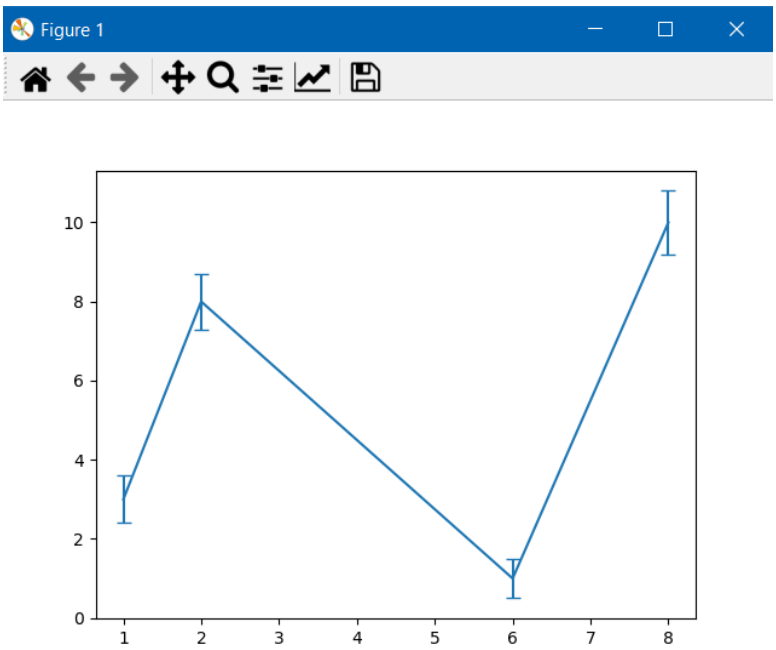


Рисунок 2.28 – Результат роботи коду із попереднього прикладу.

Тепер порівняємо результат роботи коду без завдання параметру capsize (Рисунок 2.29):

```
import numpy as np
import matplotlib.pyplot as plt
# масиви з координатами (1, 3), (2, 8), (6, 1) і (8, 10)
xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])
yerr = np.array([0.6, 0.7, 0.5, 0.8])
plt.errorbar(xpoints, ypoints, yerr)
plt.show()
```

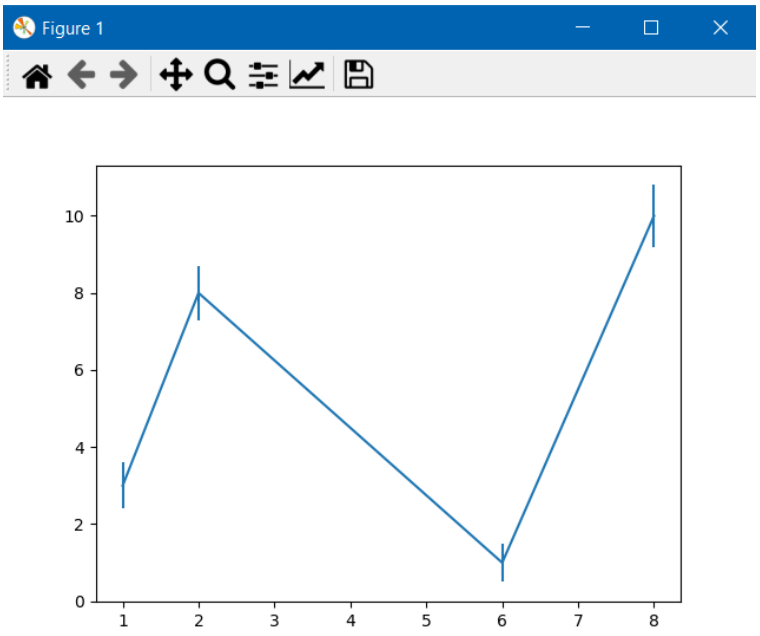


Рисунок 2.29 – Результат роботи коду із попереднього прикладу.

Наступний приклад показує можливість побудови графіка без завдання даних по осі x абсцис (Рисунок 2.30):

```
# Якщо ми не вказуємо значення координат точок по осі x,
# за замовченням вони матимуть значення
# 0, 1, 2, 3, (і т.п. в залежності від кількості координат по осі
y
# Так що, якщо взяти попередній приклад і
# не вказувати x-координати, то графік матиме вигляд:
ypoints = np.array([3, 8, 1, 10, 5, 7])
plt.plot(ypoints)
plt.show()
```

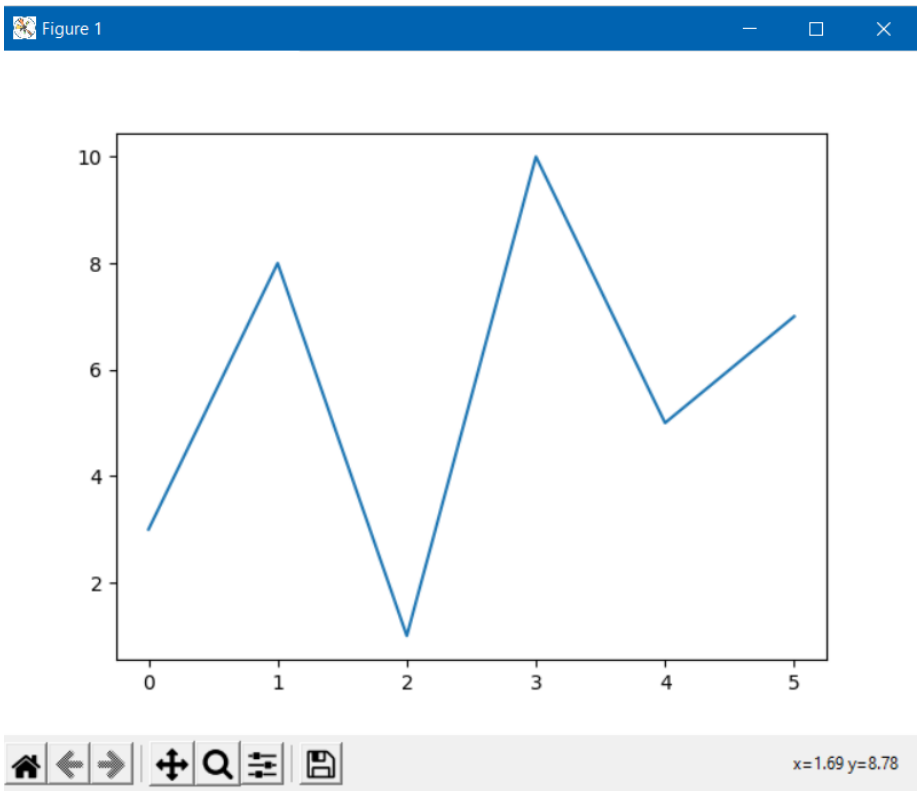


Рисунок 2.30 – Результат роботи коду із попереднього прикладу.

Matplotlib підтримує кілька типів маркерів, які вибираються за допомогою параметра marker методу plot() (Таблиця 2.1):

- незаповнені маркери (unfilled markers);
- заповнені маркери (filled markers);
- маркери, створені з символів TeX (MathText); тощо.

Наприклад, можливо задати різні типи маркерів точок: 'o' '*' '.' ',' 'x' 'X' '+' 'P' 's' 'D' 'd' 'H' 'h' '<' '>' '|' тощо (Рисунок 2.31):

```
# Помічаємо кожну точку маркером у формі кола:
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o')
plt.show()
```

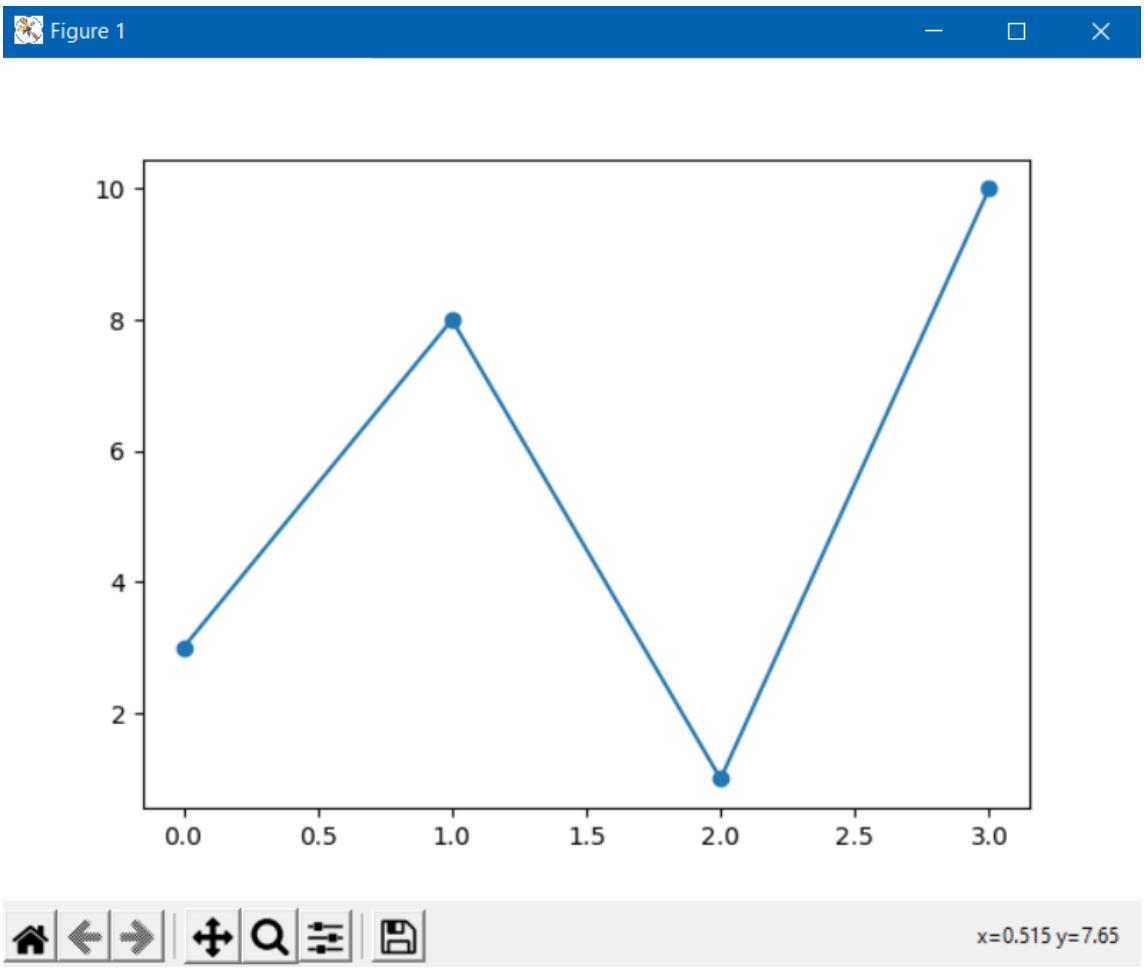


Рисунок 2.31 – Результат роботи коду із попереднього прикладу.

Тип маркерів [27] для ліній можна обирати з каталогу маркерів - Marker Reference: https://matplotlib.org/stable/gallery/lines_bars_and_markers/marker_reference.htm

Використовуються іменовані аргументи, за допомогою яких задається типи маркерів в функції plot():

- markersize (або ms) – розмір маркера
- markeredgecolor (або mec) – колір країв маркера (Таблиця 2.2)
- markerfacecolor (або mfc) – налаштування кольору маркеру всередині області, охопленої краєм маркеру тощо (Таблиця 2.2).

```
# 'o' – тип маркеру ':' – стиль лінії 'r' – колір лінії
plt.plot(ypoints, 'o:r', ms = 20, mec='g', mfc = 'b')
plt.show()
```

Результат роботи цього коду представлено на Рисунок 2.32.

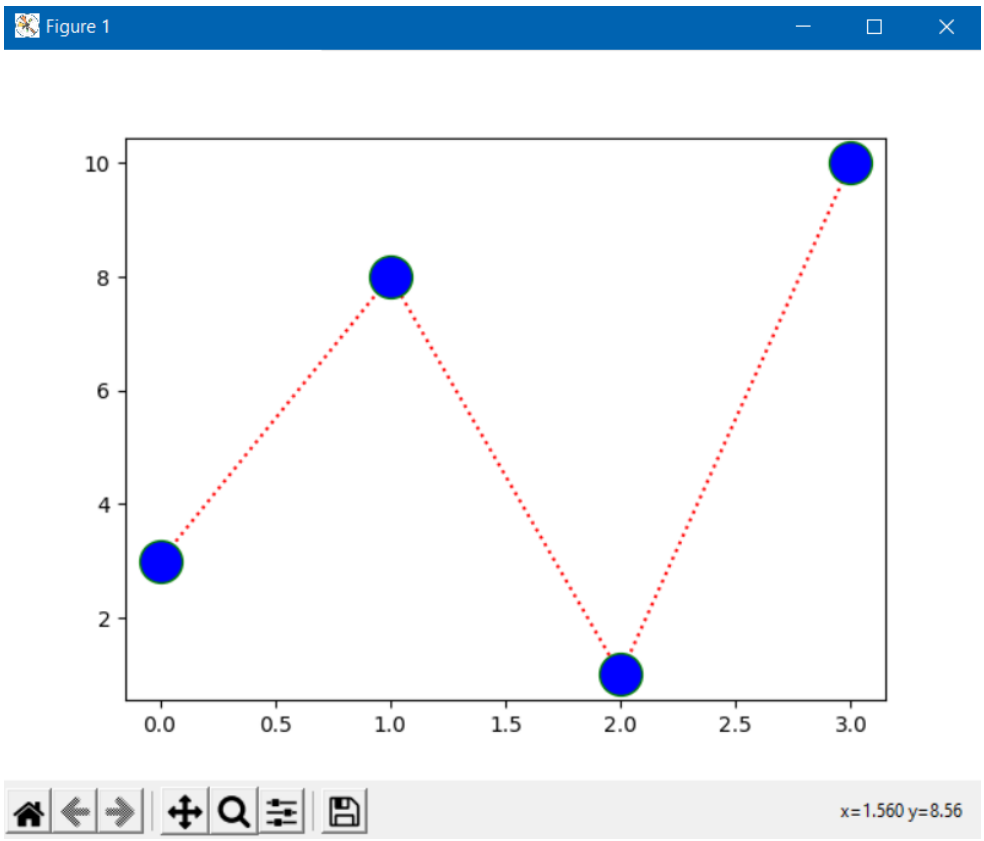


Рисунок 2.32 – Результат роботи коду із попереднього прикладу.

Таблиця 2.1. Типи маркерів для точок на графіку [27]

Маркер	Опис	
'o'	Circle	Коло
'*'	Star	Зірочка
'.'	Point	Точка
','	Pixel	Піксель
'x'	X	X
'X'	X (filled)	X (заповнений)
'+'	Plus	Плюс
'P'	Plus (filled)	Плюс (заповнений)
'D'	Diamond	Діамант
'd'	Diamond (thin)	Діамант (тонкий)
'p'	Pentagon	Пентагон
'H'	Hexagon	Шестикутник
'h'	Hexagon (thin)	Шестикутник (тонкий)

'v'	Triangle Down	Трикутник вниз
‘*'	Triangle Up	Трикутник вгору
'<'	Triangle Left	Трикутник ліворуч
'>'	Triangle Right	Трикутник праворуч
'1'	Tri Down	Tri вниз
'2'	Tri Up	Tri вгору
'3'	Tri Left	Tri ліворуч
'4'	Tri Right	Tri праворуч
'|'	Vertical line	Вертикальна лінія
'_'	Horizontal line	Горизонтальна лінія

Таблиця 2.2. Колір на графіку [27].

Синтаксис кольорів	Опис	
'r'	Red	Червоний
'g'	Green	Зелений
'b'	Blue	Синій
'c'	Cyan	Блакитний
'm'	Magenta	Пурпурний
'y'	Yellow	Жовтий
'k'	Black	Чорний
'w'	White	Білий

Найбільш поширені кольори: 'r‘, ‘g’, ‘b’, ‘c’, ‘m’, ‘y’, ‘k’, ‘w’. За посиланням https://www.w3schools.com/colors/colors_names.asp можна знайти 140 назв кольорів (Рисунок 2.33).

Окрім завдання назв кольорів можна використовувати вибір кольорів у форматі «Шістнадцяткові значення кольору» («Hexadecimal color values») (Рисунок 2.34) [33]:

```
# Можна використовувати так звані Hexadecimal color values:
plt.plot(ypoints, marker='o', c='Chartreuse', ms=20,
mec='#4CAF50', mfc='C04381')
plt.show()
```

Рисунок 2.33 – Приклад назв кольорів та зображень кольорів, запропонованих за посиланням https://www.w3schools.com/colors/colors_names.asp [32]

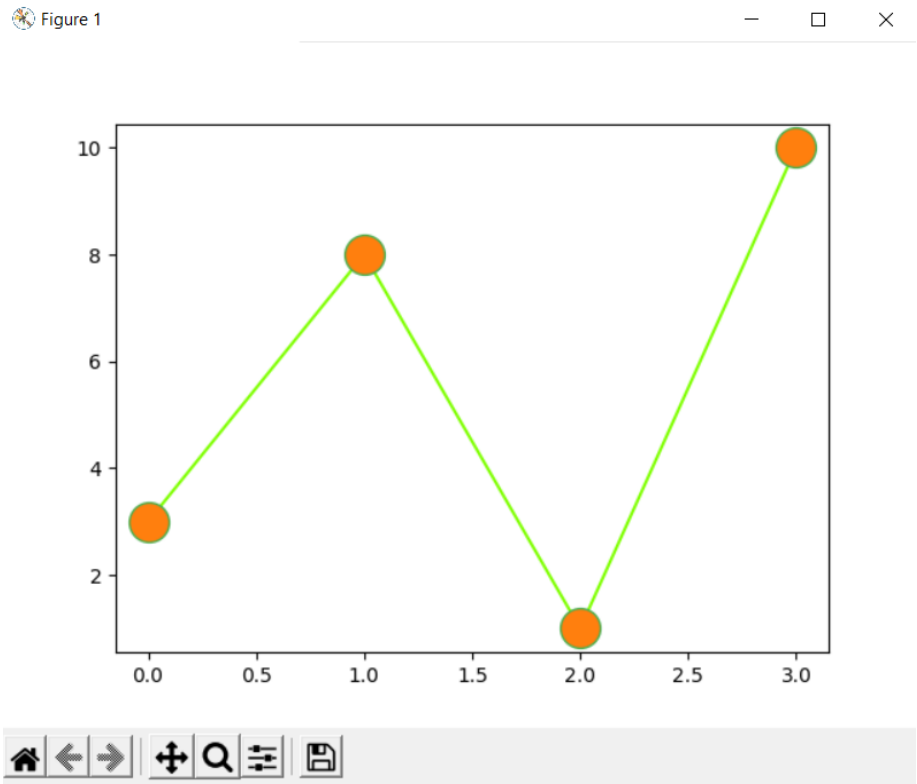


Рисунок 2.34 – Результат роботи коду із попереднього прикладу.

Online калькулятор кольору https://www.w3schools.com/colors/colors_hexadecimal.asp дозволяє здійснити перегляд кольору, заданого в форматі «Шістнадцяткові значення кольору» («Hexadecimal color values») (Рисунок 2.35).

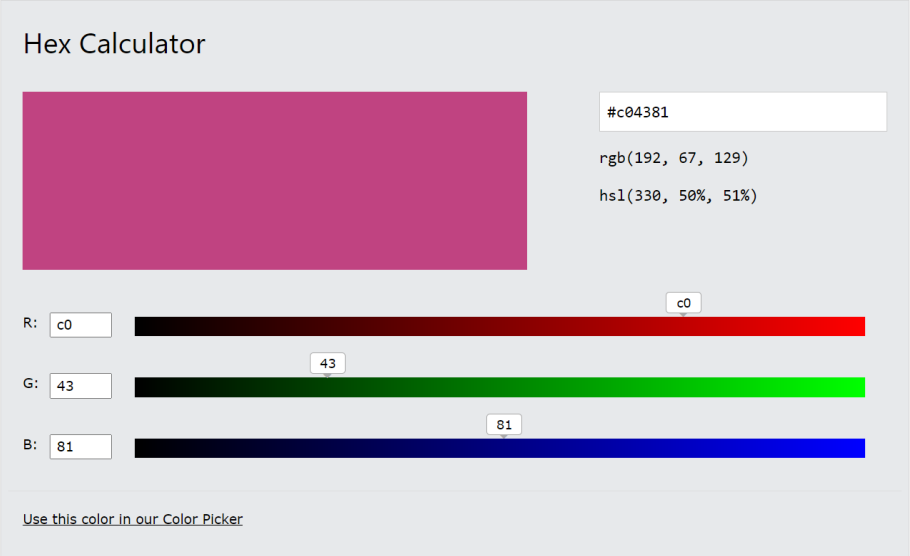


Рисунок 2.35 – Приклад перегляд кольору, заданого в форматі «Шістнадцяткові значення кольору» («Hexadecimal color values») за посиланням https://www.w3schools.com/colors/colors_hexadecimal.asp [33].

Наступний приклад показує завдання двох кривих на графіку (Рисунок 2.36):

```
# Зобразимо дві лінії з завданням функції plt.plot() для кожної з них
y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])
plt.plot(y1)
plt.plot(y2)
plt.show()
```

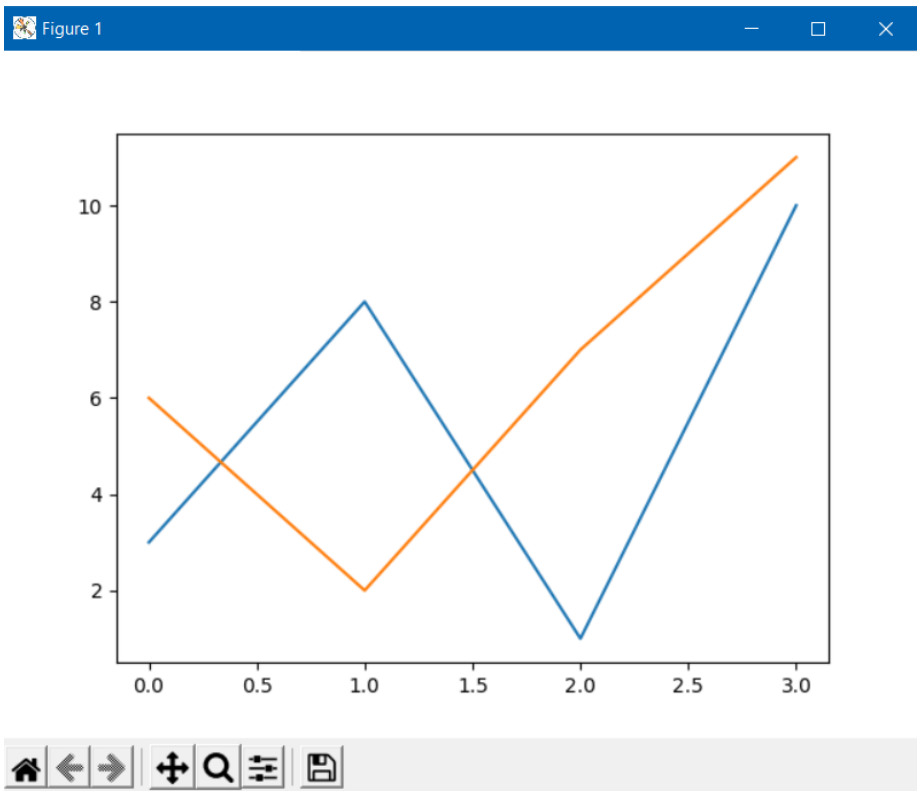


Рисунок 2.36 – Результат роботи коду із попереднього прикладу.

При оформленні графіків важливо вміти задавати підписи на графіку та його осях. Наступний приклад показує, як підписати осі та дати назву графіку (Рисунок 2.37):

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.plot(x, y)
plt.title("Dynamics of growth")
plt.xlabel("time, s")
plt.ylabel("biomass, g")
plt.show()
```

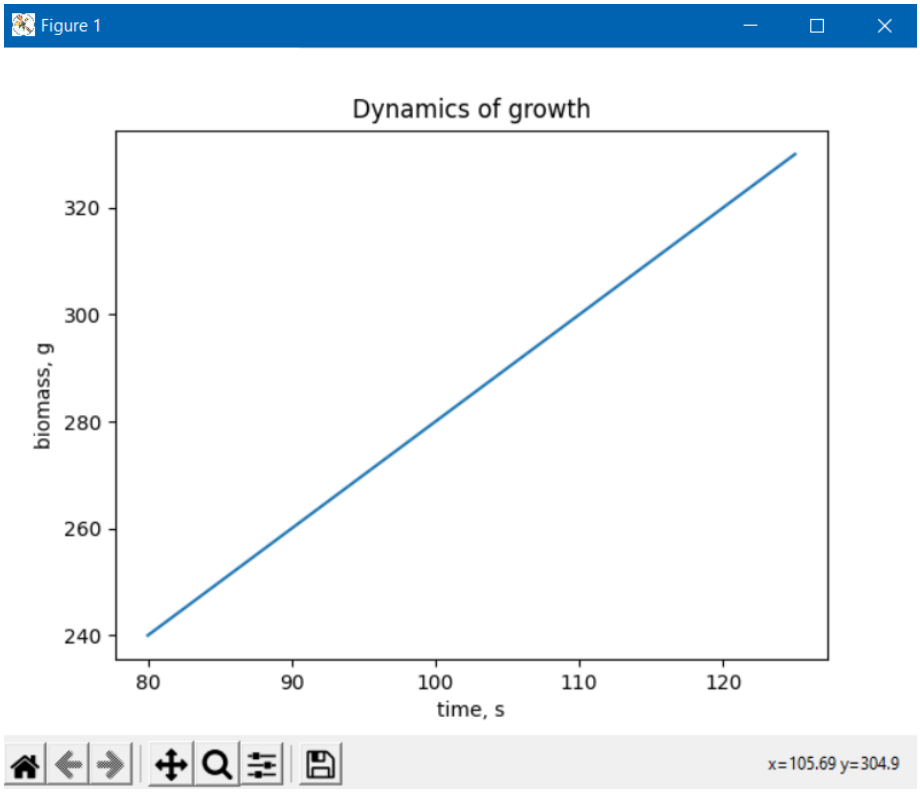


Рисунок 2.37 – Результат роботи коду із попереднього прикладу.

Наступний приклад показує, як налаштувати шрифт та колір підписів на графіку, а також, як задати сітку (Рисунок 2.38):

```
# Можна використовувати параметр fontdict для функцій
# xlabel(), ylabel() і title(), щоб налаштувати властивості назви
# графіку та підписів по осях.  З модулем Pyplot можна
#  використовувати функцію grid(), щоб додати лінії сітки на
графіку. В функції grid(), можна налаштувати властивості
# ліній сітки: grid (color='color', linestyle='linestyle',
linewidth=
# number)
import numpy as np
import matplotlib.pyplot as plt
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.plot(x, y)
```

```
font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}
# Розташовуємо назву графіку ліворуч:
plt.title("Dynamics of growth",loc ='left', fontdict = font1)
plt.xlabel("time, s", fontdict = font2)
plt.ylabel("biomass, g", fontdict = font2)
plt.grid(axis ='x')
plt.show()
```

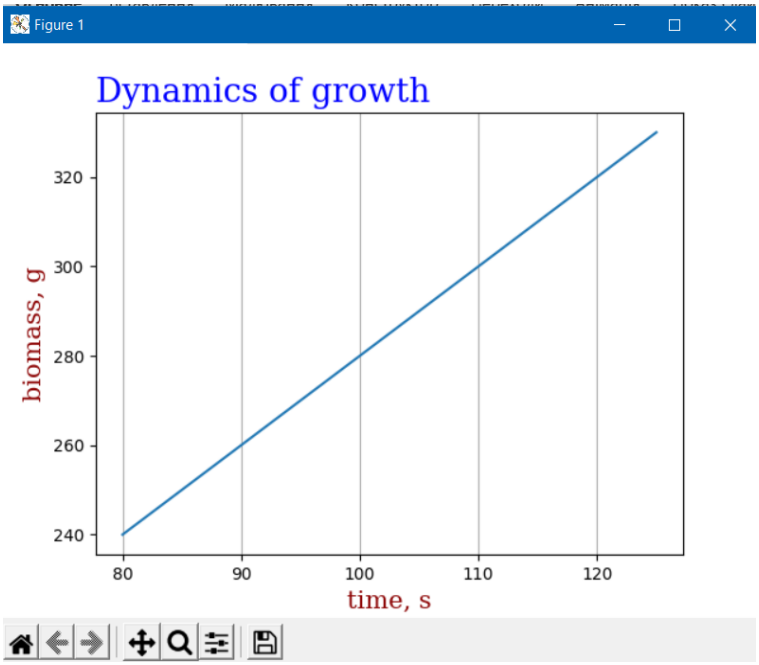


Рисунок 2.38 – Результат роботи коду із попереднього прикладу.

Наступний приклад показує, як здійснюється незалежне завдання підписів та параметрів на різних панелях (Рисунок 2.39):

```
#Графік 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
# Рисунок має 1 рядок, 2 стовпчики, і це перший графік 1.
plt.subplot(1, 2, 1)
plt.plot(x,y)
font1 = {'family':'serif','color':'blue','size':14}
font2 = {'family':'serif','color':'darkred','size':12}
plt.title("Mushroom biomass",loc ='left', fontdict = font1)
plt.xlabel("time, days", fontdict = font2)
plt.ylabel("biomass, g", fontdict = font2)
#plt.grid()
plt.grid(axis = 'y')
plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)
plt.grid(axis ='x')
#Графік 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
# Рисунок має 1 рядок, 2 стовпчики, і це другий графік 2.
plt.subplot(1, 2, 2)
plt.plot(x,y)
```

```
font1 = {'family':'serif','color':'blue','size':14}
font2 = {'family':'serif','color':'darkred','size':12}
plt.title("Tomato biomass",loc ='left', fontdict = font1)
plt.xlabel("time, days", fontdict = font2)
plt.ylabel("biomass, g", fontdict = font2)
#plt.grid()
#plt.grid(axis = 'x')
plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)
plt.grid(axis ='y')
plt.suptitle("DYNAMICS OF GROWTH")
plt.show()
```

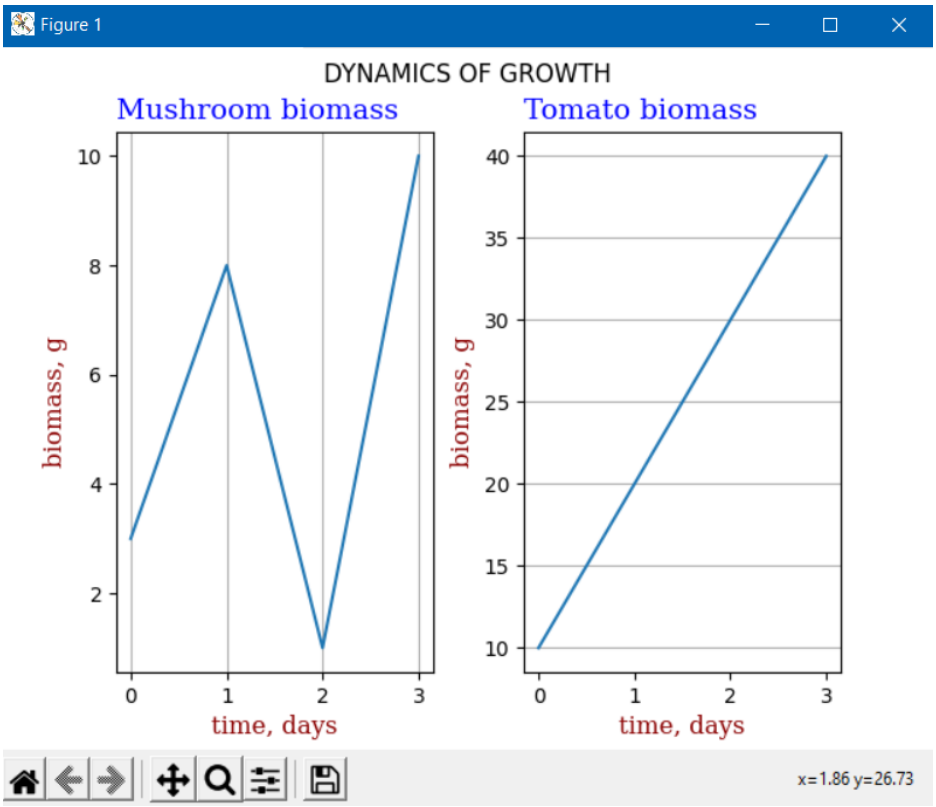


Рисунок 2.39 – Результат роботи коду із попереднього прикладу.

Cmap – matplotlib має ряд вбудованих кольорових карт [34], доступних через matplotlib.cm.get_cmap. Існують також зовнішні бібліотеки, такі як [palettable] і [colorcet], які мають багато додаткових кольорових карт. Розглянемо також приклад зображення точок не з'єднаних лініями різними кольорами, використовуючи мапи кольорів (Рисунок 2.40):

```
# scatter - зображення точок, не з'єднаних лінією, colormap -
кольорова мапа
# Налаштовуємо різні розміри для кожного маркера:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90,
100])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])
plt.scatter(x, y, c=colors, cmap='viridis', s=sizes)
plt.colorbar()
plt.show()
```

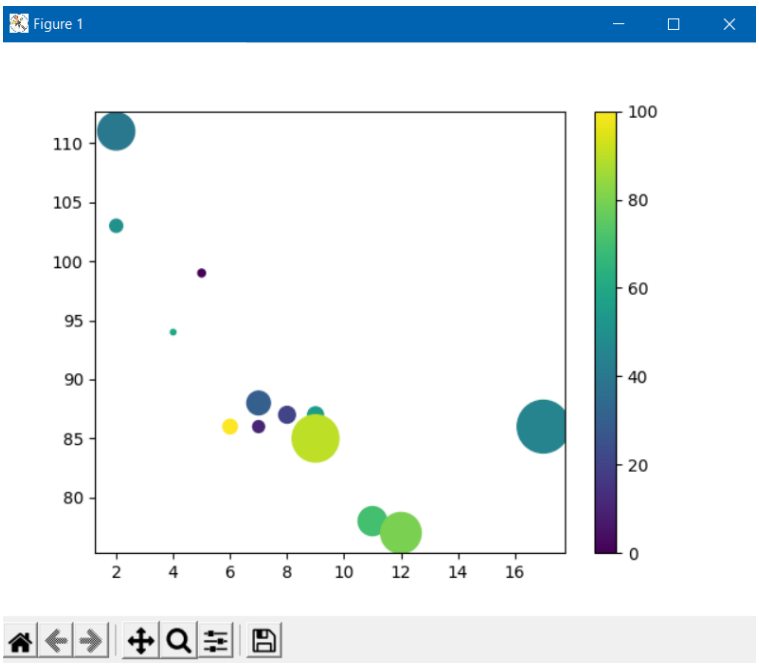


Рисунок 2.40 – Результат роботи коду із попереднього прикладу.

Вибрати мапу кольорів можна, наприклад, за посиланням (Таблиця 2.3): https://matplotlib.org/stable/tutorials/colors/colormaps.html [34]

Таблиця 2.3. Мапа кольорів – Colormap.

Назва мапи кольорів	Посилання	Обернена мапа	Посилання
Accent	Try it »	Accent_r	Try it »
Blues	Try it »	Blues_r	Try it »
BrBG	Try it »	BrBG_r	Try it »
BuGn	Try it »	BuGn_r	Try it »
BuPu	Try it »	BuPu_r	Try it »
CMRmap	Try it »	CMRmap_r	Try it »
Dark2	Try it »	Dark2_r	Try it »
GnBu	Try it »	GnBu_r	Try it »
Greens	Try it »	Greens_r	Try it »
Greys	Try it »	Greys_r	Try it »

Функція bar() модулю pyplot пакету matplotlib використовується для побудови вертикальної діаграми.

Функція plt.bar(x, height, width=0.8, bottom=None, align='center', data=None, **kwargs) має такі найбільш використовувані параметри:

- x (float або масив) – координати x стовпчиків;
- height (float або масив) – висота(и) стовпчиків;
- width (float або масив) – ширина(и) стовпчиків за замовчуванням 0.8;
- bottom (float або масив) – координата(и) у основи стовпчиків, за замовчуванням 0;
- align {'center', 'edge'} вирівнювання стовпчиків за координатами x , за замовчуванням 'center‘;

а також низку інших параметрів:

- color – кольори стовпчиків;
- edgecolor – кольори країв стовпчиків;
- linewidth – ширина краю(їв) стовпчиків;
- tick_label (str або список str) – позначки рисок на осях;
- xerr, yerr (float) – значення похибок;
- ecolor (колір або список кольорів) – колір лінії похибки, за замовчуванням: 'чорний';
- capsize (float)– ширина позначки похибки, за замовчуванням: 0.0.

Наступний приклад показує метод побудови вертикальної діаграми (Рисунок 2.41):

```
# З модулем Pyplot, можна використовувати функцію bar(),
# щоб зобразити вертикальну діаграму:
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x,y)
plt.show()
```

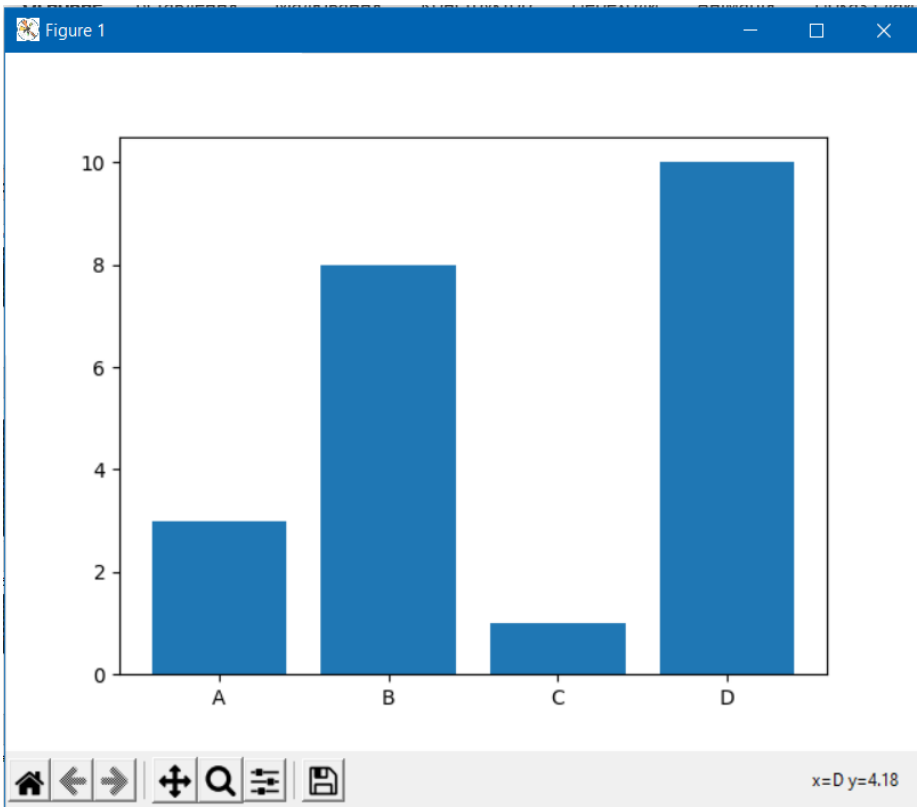


Рисунок 2.41 – Результат роботи коду із попереднього прикладу.

Метод barh() використовується для побудови горизонтальної діаграми (Рисунок 2.42):

```
# Зображуємо горизонтальну діаграму, яка складається з 4-ох
стовпчиків:
# Функція побудови горизонтальної діаграми barh () приймає ключове
слово-
# аргумент – height для налаштування висоти стовпчиків:
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.barh(x, y, height=0.1)
#Функція побудови вертикальної діаграми bar()
# приймає ключове слово-аргумент – width
# для налаштування ширини стовпчиків:
plt.bar(x,y, width=0.1)
plt.show()
```

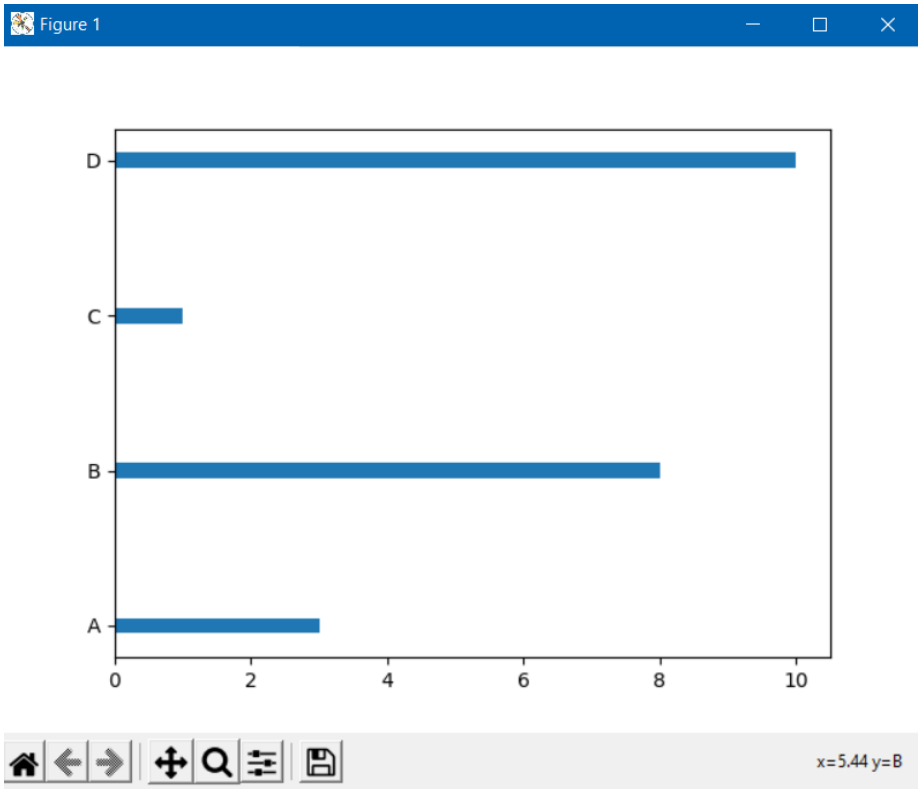


Рисунок 2.42 – Результат роботи коду із попереднього прикладу.

З модулем Pyplot, можна використовувати функцію pie () для зображення секторної діаграми (Рисунок 2.43):

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
y = np.array([35, 25, 25, 15])
mylabels = ["% A", "% G", "% C", "% T"]
mycolors = ["black", "hotpink", "b", "#4CAF50"]
plt.pie(y, labels=mylabels, colors=mycolors)
plt.legend(title="% Nucleotides")
fig.savefig("fig.png", orientation='landscape', dpi=300)
plt.show()                    # dpi - dots per inch
```

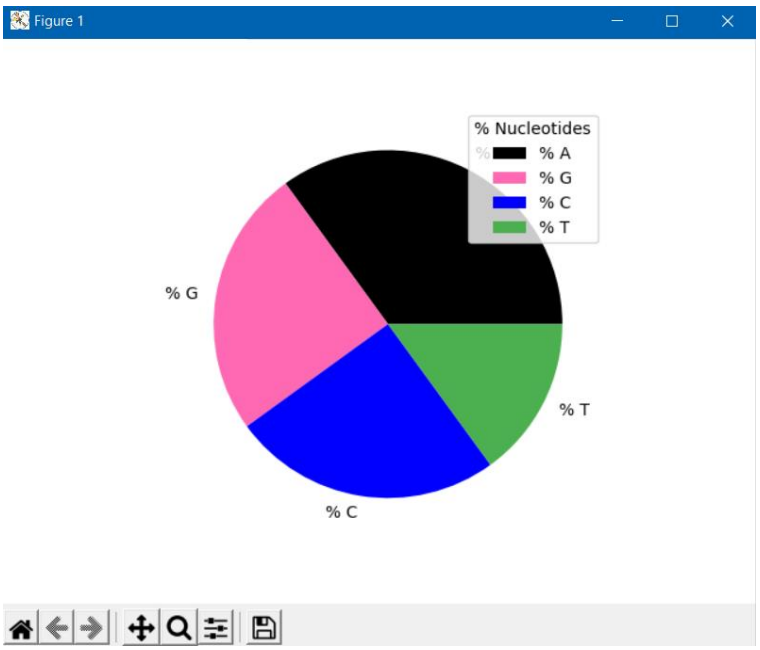


Рисунок 2.43 – Результат роботи коду із попереднього прикладу.

В попередньому прикладі ми використали метод Figure.savefig. Розглянемо параметри методу Figure.savefig:

- fname – рядок, що містить шлях до файлу або схожий на файл об'єкт python. Формат рисунка визначається по розширенню імені файлу, наприклад: PDF для .pdf і PNG для .png;
- dpi – роздільна здатність рисунка в пікселях на дюйм; за замовчуванням 100 dpi , але може налаштовувати;
- facecolor, edgecolor – колір фону рисунку поза областю, зайнятою графіками, за замовчуванням – w (білий);
- format – явно заданий формат файлу (png, pdf, svg, ps, eps тощо);
- bbox_inches – налаштовує частину рисунку, яку необхідно зберігати.

Завдання та запитання до підрозділу «Пакет matplotlib для побудови графіків та діаграм»

1. Напишіть код, який робить наступне: генерує масив даних, які імітують експериментальні дані результатів вимірювання розмірів наночастинок; зображує гістограму розподілу наночастинок за розмірами.
2. Які методи використовуються для вставки підписів на осях графіка?
3. Який метод використовуються для вставки назви графіка?
4. Які методи використовуються для вставки легенди графіка?
5. Як налаштувати тип, розмір та колір шрифту підписів на графіку?
6. Який метод використовується для зображення похибки вимірювань на графіку або на діаграмі?
7. Які методи побудови графіків та діаграм пакету matplotlib Ви знаєте?

2.4. Пакети python для задач регресійного аналізу

Провідні фармацевтичні компанії світу для створення передової терапії та розвитку персоналізованої медицини використовують машинне навчання та штучний інтелект для пошуку нових молекул та їх комбінацій: Roche (53bn), Pfizer (49.7bn), Johnson & Johnson (42.2bn), Merck (41.8bn), Novartis (37.7bn), AbbVie (33.3bn), Takeda (31.1bn), Bristol–Myers Squibb (26.1bn).

Бібліотеки мови python numpy, scipy [25, 35], scikit-learn (sklearn) [36, 37], matplotlib, pandas використовуються для інженерних та наукових досліджень, в тому числі для задач регресійного аналізу та машинного навчання.

Як вже зазначалося, numpy (numeric python) – це бібліотека на мові python, що допомагає підтримувати більшість різноманітних масивів і матриць, разом з великою бібліотекою високорівневих (і дуже швидких) математичних функцій для операцій з цими масивами.

Numpy – досить низькорівневий інструмент, схожий на MATLAB. Pandas, з іншого боку, надає багаті функціональні можливості для аналізу часових рядів, аналізу даних, зручну для використання статистику, методи злиття і об'єднання даних, а також безліч інших зручностей. Бібліотека стала дуже популярною в останні роки в фінансових додатках. Бібліотека pandas спрощує життя аналітикам: де раніше використовувалося 10 рядків коду тепер вистачить одного.

Scipy (scientific python) – це відкрита бібліотека високоякісних інструментів в науці і для інженерних розробок для мови програмування python, містить модулі для оптимізації, інтегрування, обробки сигналів, обробки зображень, генетичних алгоритмів, розв'язування звичайних диференціальних рівнянь тощо [25, 35]. Numpy і scipy – це бібліотеки python, які використовуються для математичного та числового аналізу. Якщо Ви імпортуєте scipy, то numpy окремо імпортувати не потрібно.

Scikit-learn (sklearn) – це бібліотека машинного навчання з відкритим вихідним кодом [37]. Пакет sklearn також надає різні інструменти для побудови моделей, попередньої обробки даних, вибору та оцінки якості моделі тощо. Пакет sklearn побудований на numpy, scipy та matplotlib.

Посібник користувача sklearn https://scikit-learn.org/stable/user_guide.html [36]. Сайт пакету sklearn https://scikit-learn.org/stable/ [37].

Переваги використання пакету sklearn :

- Прості та ефективні інструменти для прогнозування та аналізу даних.
- Відкритий вихідний код розповсюдження програмного забезпечення.
- Побудований на numpy, scipy та matplotlib.

Scikit-learn (sklearn) – це найкорисніший та надійний пакет для машинного навчання на python. Він надає вибір ефективних інструментів для машинного навчання та статистичного моделювання, включаючи класифікацію, регресію, кластеризацію та зменшення розмірності через узгоджений інтерфейс у python. Машинне навчання – множина математичних, статистичних та обчислювальних методів штучного інтелекту для розробки алгоритмів, здатних вирішити задачу не прямим способом, а на основі пошуку закономірностей в різноманітних вхідних даних, системи здатні навчатися.

Пакет pandas – це програмна бібліотека, написана для мови програмування python для структурування даних та їхнього аналізу. Pandas надає високорівневі інструменти для роботи з даними, побудовані на основі numpy. Pandas, зокрема, пропонує структури даних та операції для маніпулювання чисельними таблицями та часовими рядами. Пакет pandas є вільним програмним забезпеченням. Його назва походить від терміну «панельні дані» (англ. panel data), який в економетрії позначає багатовимірні структуровані набори даних. Економетрія – наука, яка вивчає кількісні та якісні економічні взаємозв'язки з використанням математичних і статистичних методів та моделей.

Алгоритми машинного навчання тісно пов’язані з обчислювальною статистикою, моделі будуються на основі вибіркових даних, відомих як навчальні дані, застосовують статистичні прийоми для надання комп'ютерам здатності «навчатися» з даних (тобто, поступово покращувати продуктивність у певній задачі), щоб робити прогнози або приймати рішення без їх явного програмування.

Алгоритми машинного навчання використовуються в широкому спектрі додатків, наприклад у медицині, фільтрації електронної пошти, розпізнаванні мовлення, комп’ютерному зорі, де важко або неможливо розробити звичайні алгоритми для виконання

необхідних завдань. Тобто машинне навчання – це програмна модель, що шляхом навчання апроксимує дані знаходячи в них кореляцію, після навчання може використовуватись для передбачення результату змодельованої системи.

Інструменти для машинного навчання та статистичного моделювання в пакеті scikit-learn (sklearn):

- Класифікація даних – визначення, до якої категорії даних належить об’єкт (застосування: виявлення спаму, розпізнавання зображень, алгоритми: метод k-найближчих сусідів, випадковий ліс тощо. Алгоритм k-найближчих сусідів знаходить відстані між запитом і всіма прикладами даних, вибираючи певну кількість прикладів (k), найбільш близьких до запиту, потім голосує за мітку, що найчастіше зустрічається (у разі задачі класифікації) або усереднює мітки (у випадку задачі регресії). Випадковий ліс, як і випливає з назви, складається з великої кількості окремих дерев рішень, які працюють як ансамбль методів. Кожне дерево у випадковому лісі повертає результат прогнозу, і результат із найбільшою кількістю голосів стає прогнозом лісу.
- Кластеризація даних, кластерний аналіз або кластеризація – це завдання групування набору об’єктів таким чином, щоб об’єкти в одній групі (яка називається кластером) були більш схожі (в певному сенсі) один на одного, ніж об’єкти в інших групах (кластерах). Задача кластеризації схожа із задачею класифікації, є її логічним продовженням, але відмінність її в тому, що класи досліджуваного набору даних заздалегідь не зумовлені. Синонімами терміну «кластеризацію» є «автоматична класифікація», «навчання без вчителя» і «таксономія». Кластеризація даних використовує методи K-середніх, зсуву середнього значення, спектральної кластеризації, ієрархічної кластеризації тощо. Метод K-середніх – найбільш відомий алгоритм кластеризації. Цей алгоритм будує задане число кластерів, розташованих якнайдалі один від одного. Метод зсуву середнього значення – алгоритм, який виконує пошук області точок даних з найбільшою щільністю. Мета алгоритму полягає у пошуку центральних точок кожного кластеру. Методи спектральної кластеризації використовують спектр (тобто значення матриці подібності даних) для виконання зменшення розмірності перед кластеризацією в меншій кількості вимірювань. Матриця подібності надається як вхідна інформація та складається з кількісної оцінки відносної подібності кожної пари точок у наборі даних. На відміну від інших методів, таких як метод K-середніх, що шукають щільні, компактні, опуклі кластери, спектральна кластеризація може знаходити кластери довільної форми. Основним результатом ієрархічної кластеризації є дендрограма (дерево), яка показує ієрархічну залежність між кластерами.
- *Кореляційний* та *регресійний аналіз* – в теорії ймовірностей і математичній статистиці – залежність, що встановлює відповідність між випадковими змінними, тобто математичний вираз, що відображає зв'язок між залежною змінною *y* і незалежними змінними *x* за умови, що цей зв'язок має статистичну значимість. Застосування: реакція на лікарські засоби, ціни на акції, споживання електроенергії у житловому будинку з даних температури, часу доби та кількості мешканців тощо. Алгоритми: найближчі сусіди, випадковий ліс.
- У цьому розділі ми вивчимо, що таке набори даних, як обчислити важливі параметри на основі наборів даних, дізнаємось, як використовувати різні модулі P5ython, щоб отримати відповіді, які нам потрібні, навчимося створювати функції, здатні передбачити результат на основі наборів даних. Набір даних – це може бути що завгодно, від масиву до бази даних.

2.4.1. Регресійний та кореляційний аналізи і задачі апроксимації в інженерних та наукових дослідженнях

В інженерній та науковій роботі часто виникають такі задачі:

- Описати експериментальні дані заданою функцією (задача апроксимації). Для розв'язання цієї задачі використовуються методи регресійного аналізу. Різниця між регресією та апроксимацією полягає в тому, що регресія – це інструмент, а апроксимація – задача. Те саме, як цвях, який треба забити, – це апроксимація, а молоток, яким можна це зробити – це регресія. При цьому інструмент – регресію можна застосовувати як для задачі апроксимації, так для задач інтерполяції, екстраполяції та інших задач.
- Встановити, чи існує зв'язок між двома величинами. Для розв'язання цієї задачі використовуються методи кореляційного аналізу.
- Таким чином, в інженерній та науковій роботі часто доводиться зустрічатися із задачею апроксимації. *Апроксимацією* називається процес підбору емпіричної формули *φ(x)* для встановленої з експериментальних даних функціональної залежності *y= f(x).* Емпіричні формули використовують для аналітичного подання експериментальних даних.

Сформулюємо задачу функціональної апроксимації для випадку однієї незалежної змінної. Нехай є деякі дані, отримані практичним шляхом (під час експерименту, спостереження тощо), які можна представити парами чисел *(x; y)*. На основі цих даних потрібно підібрати функцію *y =φ(x)*, яка щонайкраще описувала б експериментальну залежність між змінними *x* і *y* та по можливості точно відображала б загальну тенденцію залежності між ними.

Звичайно задача апроксимації розпадається на дві частини. Спочатку встановлюють експериментальні дані для залежності *y =f(x)* і, відповідно, вид емпіричної формули *y =φ(x)* (поліноміальна, експоненціальна, логарифмічна тощо). Після цього визначаються чисельні значення невідомих параметрів обраної емпіричної формули, для яких наближення до заданого в експерименті набору даних *y =f(x)* виявляється найкращим. При відсутності теоретичних міркувань при підборі виду формули *y =φ(x)*, зазвичай вибирають функціональну залежність з числа відомих, порівнюючи їхні графіки із графіком заданих в експерименті експериментальних даних *y =f(x)*. Після вибору виду формули визначають її параметри. Для найкращого вибору параметрів задають міру наближення апроксимації до експериментальних даних. У багатьох випадках, особливо якщо функція *f(x)* задана графіком або таблицею (на дискретній множині точок), для оцінки ступеня наближення розглядають різниці *f(x)-φ(x)* для точок $x_0, x_1, \ldots, x_к$. Існують різні ступені наближення та, відповідно, методи розв'язання цієї задачі, зокрема метод найменших квадратів. При цьому функція *φ (x)* вважається найкращим наближенням до експериментальної залежності *f(x)*, якщо для неї сума квадратів відхилень $\varphi(x_i)$ від відповідних значень $f(x_i)$ є мінімальною:

$$Z = \sum_{i=0}^{n} [f(x_i) - \varphi(x_i)]^2 \to min,$$

тобто має найменше значення в порівнянні з іншими функціями, з числа яких вибирається шукане наближення.

Важливо розуміти, що для низки задач існують теоретичні міркування при підборі виду формули *φ(x)* на основі літературних даних про відомі теоретичні моделі поведінки досліджуваної системи.

Кореляційний та *регресійний аналіз* – це статистичні дослідження залежності між випадковими величинами. Різниця між цими двома методами у меті: для кореляційного аналізу важливо виявити, чи існує зв'язок між двома величинами, а для регресійного – аналітичний вираз (функцію регресії) цієї залежності. Таким чином, регресійний аналіз створює математичну модель зв'язку, а кореляційний аналіз використовується, коли дослідник хоче знати, чи є співвідношення досліджуваних змінних чи ні, якщо так, то яка сила їхньої асоціації. В Таблиці 2.4 наведено порівняння понять кореляція та регресія.

Регресійний аналіз – розділ математичної статистики, присвячений методам аналізу залежності однієї величини від іншої. Регресійний аналіз не з'ясовує сутність зв'язку, а займається пошуком моделі цього зв'язку, вираженої у функції регресії.

Регресійний аналіз використовується в тому випадку, якщо відношення між змінними можуть бути виражені кількісно у вигляді деякої комбінації цих змінних. Отримана комбінація використовується для передбачення значення, що може приймати цільова (залежна) змінна, яка обчислюється на заданому наборі значень вхідних (незалежних) змінних. У найпростішому випадку для цього використовуються стандартні статистичні методи, такі як лінійна регресія, яка буде описана далі.

2.4.2. Порівняння понять кореляція та регресія

Таблиця 2.4. Порівняння понять кореляція та регресія

Ознака для порівняння	**Кореляція**	**Регресія**
Значення	Кореляція - це статистичний показник, який визначає взаємозв'язок або асоціацію двох змінних.	Регресія описує, як незалежна змінна чисельно пов'язана з залежною змінною.
Використання	Для представлення лінійної залежності між двома змінними.	Щоб підібрати кращу функціональну залежність і оцінити одну змінну на основі іншої змінної.
Сенс	Коефіцієнт кореляції вказує на ступінь наявності лінійного взаємозв’язку між двома змінними.	Регресія вказує на вплив зміни однієї величини іншу змінну, що оцінюється.
Мета	Знайти числове значення, що виражає ступінь лінійного взаємозв'язку між змінними.	Оцінити значення однієї величини на основі значень іншої.

2.4.3. Типи та набори даних. Побудова функції розподілу для наборів даних. Нормальний розподіл

Для аналізу даних важливо знати, з яким типом даних ми маємо справу. Знаючи тип даних та джерела даних, Ви зможете знати, які методи використовувати при їх аналізі.

Можна розділити типи даних на три основні категорії:

- кількісні (числові) дані (ще їх називають варіаційними)
- категоричні
- порядкові.

Кількісні дані можна розділити на дві категорії: дискретні та неперервні.

Різниця між дискретною змінною та неперервною змінною полягає в тому, що область дискретної змінної є зліченною, тоді як область неперервної змінної складається з усіх дійсних значень у межах певного діапазону.

Це значить, що дискретні дані – дані, які можуть приймати строго визначений набір значень, наприклад такий, що обмежений цілими числами. Приклад: кількість студентів на лекції.

Неперервні (безперервні) дані – дані представлені будь-якими значеннями, такі як довжина, вага, температура, вік – можуть бути виміряні достатньо точно, і не мають мінімальної неподільної одиниці виміру. Наприклад, вага може бути виміряна з точністю до грама, а може – й до мікрограма, а вік – з точністю до днів, годин, хвилин, секунд.

Кількісні дані можна представити двома способами: у вигляді неперервної змінної – тоді, наприклад, вказується точний вік пацієнта (60,3 року), а також у вигляді дискретної змінної – вказується тільки кількість повних років (60 років).

Категоріальні (категорійні) дані стосуються форми інформації, яку можна зберігати та ідентифікувати на основі їхніх імен або міток. Це тип якісних даних, які можна згрупувати в категорії замість того, щоб вимірювати чисельно. Категорійні дані – це значення, які неможливо порівняти між собою. Приклад: значення кольору або будь-які значення так/ні.

Порядкові дані схожі на категорійні дані, але їх можна порівняти один з одним. Приклад: оцінки студентів, де А краще, ніж В тощо. Таким чином, порядкові дані – це тип категоріальних даних, в яких категорії мають природний порядок або ранжування.

Будь-яка кількісна характеристика, яка в результаті випадкового експерименту може прийняти одне з деякої кількості значень, – це випадкова величина. Основна ідея полягає в тому, що у нас є реальне джерело даних. Ми називаємо це джерело даних розподілом. Ми можемо збирати дані з цього джерела в будь-якому обсязі. Дані, які ми збираємо, називаються випадковою величиною розподілу. Кожне можливе значення випадкової величини ми називаємо результатом. В процесі збирання (тобто вимірювання конкретних значень випадкової величини з розподілу) ми отримуємо набір даних.

Набір значень, який збирається, називається вибіркою. Набір значень, які складають вибірку, часто називають «даними». У інженерних та наукових застосуваннях випадковою величиною зазвичай є число, яке є результатом вимірювання властивостей досліджуваної системи. Часто ми називаємо процес отримання значення для випадкової змінної «проведенням експерименту», «виконанням тесту» або «проведенням випробування».

Функція розподілу та щільність ймовірності розподілу найбільш повно та наочно описують діапазон зміни випадкової величини. Для побудови залежностей функції розподілу та густини ймовірності необхідно провести прямі багаторазові вимірювання заданої величини X, внаслідок чого отримати набір значень (вибірку) $(x_1, x_2, \cdots x_n)$ для n вимірювань.

Область значень розіб'ємо на рівні інтервали та визначимо кількість вимірювань, що потрапили в той чи інший інтервал. Позначимо через $(m_1, m_2, \cdots m_k)$ - кількість вимірювань (абсолютну частоту), що потрапили, відповідно до першого, другого, ..., k-ого інтервалу, стільки ж маємо стовпчиків діаграми - всього k стовпчиків. При цьому кількість вимірювань n >> k. Відносна частота (ймовірність розподілу): $P_i = \frac{m_i}{n}$, де i – порядковий номер інтервалу. Щільність ймовірності або щільність відносної частоти: $f(x_i) = \frac{P_i}{\Delta x} = \frac{m_i}{\Delta x \cdot n}$ (Рисунок 2.44).

Функцію $f(x_i)$ називають також диференціальним законом розподілу випадкової величини X, або диференціальною функцією розподілу. Густиною (щільністю) імовірностей неперервної випадкової величини X є перша похідна від інтегральної функції розподілу ймовірностей F(x). Побудова диференціальної функції розподілу для наборів даних

$$f(x_i) = \frac{m_i}{\Delta x \cdot n}$$

Відкладаючи по осі абсцис інтервали Δx, а осі ординат – значення $f(x_i)$, будується гістограма щільності ймовірності розподілу.

За умови проведення великої кількості випробувань ми визначимо значення:

- ймовірності того, що величина X може набувати певних значень у кожному інтервалі $(x_i, x_i + \Delta x)$

$$P(x_i) = \frac{m_i}{n}$$

тобто

$$P(x) = \frac{m}{n}$$

- щільність ймовірності (розподілу) – ймовірність попадання випадкової величини в заданий інтервал

$$f(x_i) = \frac{m_i}{\Delta x \cdot n}$$

тобто

$$f(x) = \frac{m}{\Delta x \cdot n} = \frac{P(x)}{\Delta x}$$

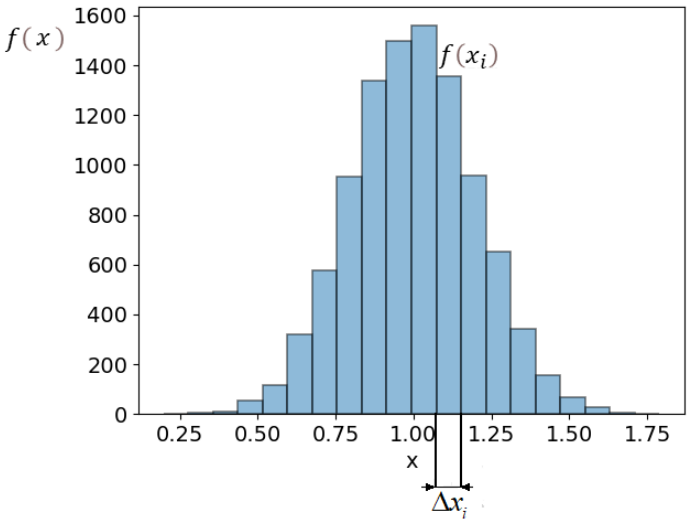


Рисунок 2.44 – Схематичне зображення густини (щільності) функції розподілу

Для знаходження остаточної залежності щільності ймовірності необхідно збільшити кількість інтервалів. При цьому довжина інтервалу Δx прямує до нуля, а гістограма перетворюється на плавну криву (Рисунок 2.45).

Найбільш відомим з усіх розподілів, які застосовують найбільше за інші для аналізу експериментальних даних є нормальний розподіл (розподіл Гауса). Оскільки нормальний розподіл гарно описує багато природних явищ, то він став стандартом відліку для багатьох ймовірнісних / статистичних задач.

Це розподіл, у якому всі три міри центральної тенденції збігаються – тобто середнє дорівнює медіані і дорівнює моді.

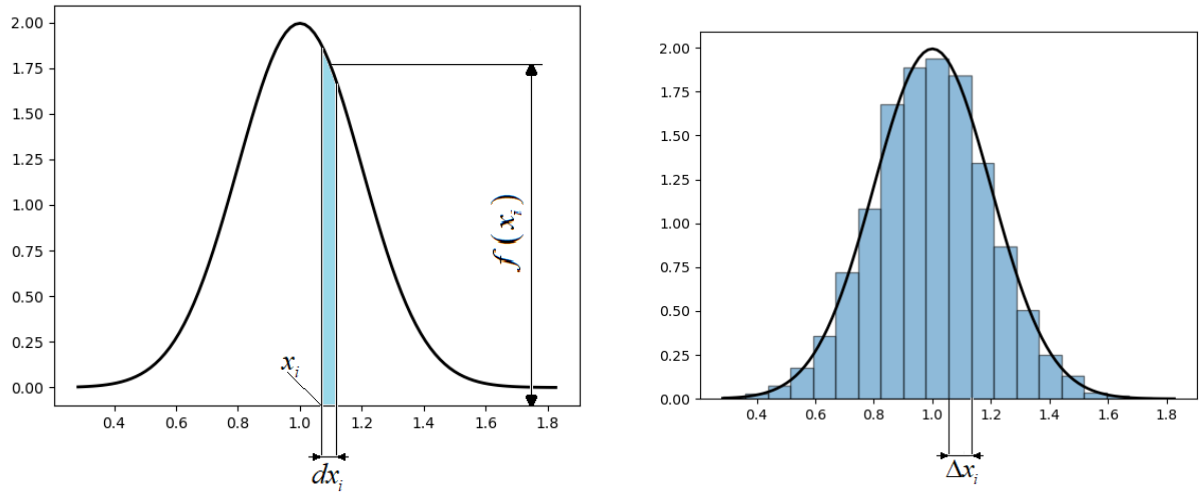


Рисунок 2.45 – Схема побудови густини функції розподілу

Нормальний розподіл також називають «дзвоноподібним» (bell-shaped curve) – адже графік нормального розподілу подібний на форму дзвона у профіль (Рисунок 2.46).

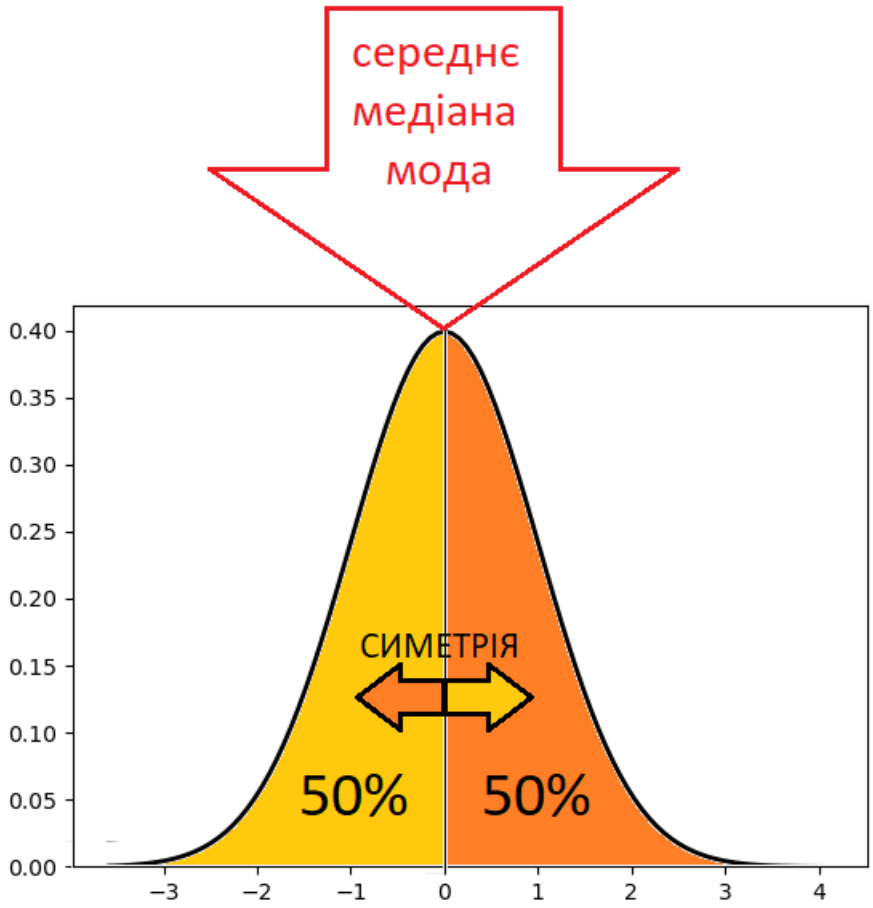


Рисунок 2.46 – Схематичне зображення нормального розподілу.

У теорії ймовірностей найбільш відомим є нормальний розподіл даних, або розподіл даних Гауса, названого так на честь математика Карла Фрідріха Гауса, який запропонував для опису таку формулу:

$$G(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right),$$

де μ – математичне очікування (середнє значення), σ^2 - дисперсія, σ – стандартне відхилення, при $\sigma^2 = 1$ – це стандартний нормальний розподіл. Функція $G(x, \mu, \sigma)$ представляє собою диференціальну функцію розподілу, визначену вище.

2.4.4. Створення та візуалізація випадкових наборів даних на мові програмування python

У машинному навчанні набори даних можуть містити тисячі або навіть мільйони значень. Ви можете не мати даних реального дослідження під час тестування алгоритму, в такому випадку використовуються випадково створені дані. Як ми дізналися в попередньому розділі, модуль numpy може нам у цьому допомогти.

Створимо масив, заповнений 1000 випадковими числами з нормального розподілу даних: середнє значення у масиві становитиме 5,0; стандартне відхиленням 1,0.

У реальних сучасних задачах набори даних включають сотні і навіть тисячі вимірювань, але збирати реальні дані може бути важко, принаймні на стадії вивчення методів python.

Як ми можемо отримати набори великих даних? Для створення великого набору даних для їх тестування використовуються функції пакету numpy такі як numpy.random.normal(), numpy.random.uniform(), numpy.empty() тощо.

Наступний код будує гістограму типового нормального розподілу даних (Рисунок 2.47):

```
import numpy as np
import matplotlib.pyplot as plt
x = np.random.normal(loc =6.0, scale = 1.0, size =100000)
plt.hist(x, 100)
plt.show()
```

У функції random.normal() пакету numpy є такі параметри:
loc – середнє значення розподілу випадкової величини ($\bar{x}$), за замовчуванням 0.0;
scale – стандартне відхилення (σ), за замовчуванням 1.0;
size (розмір) – визначає форму повернутого масиву, за замовчуванням 1.

По осі Y (Рисунок 2.47) відображається скільки разів повториться відповідне значення X із набору випадкових значень, тобто щільність ймовірності розподілу випадкової величини.

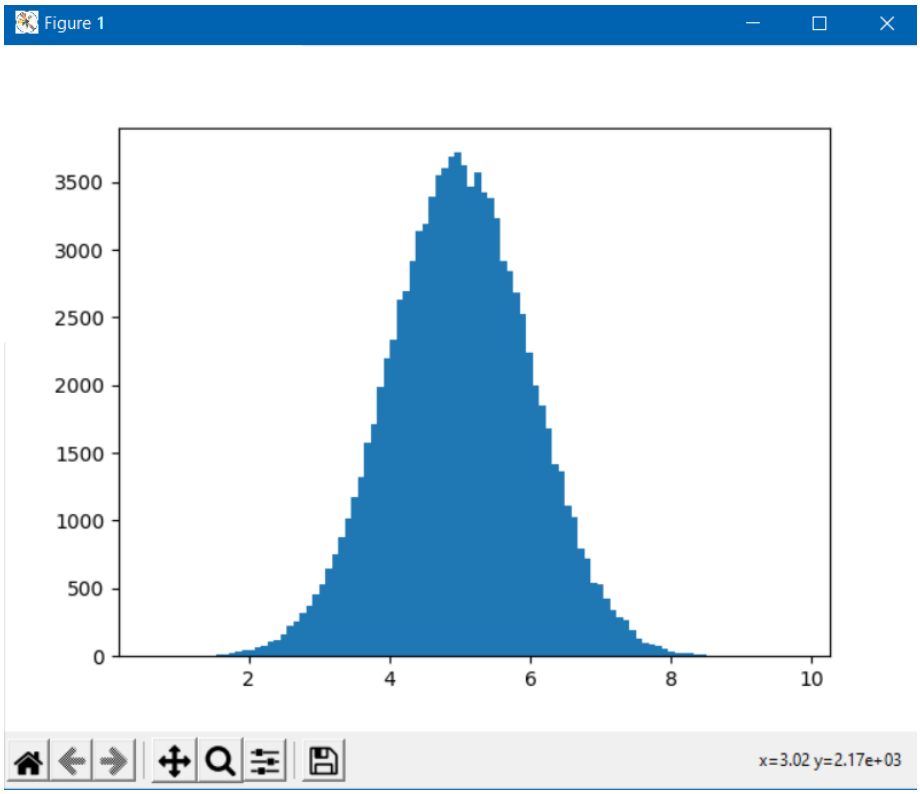


Рисунок 2.47 – Гістограма типового нормального розподілу даних.

Створимо два масиви, заповнені 1000 випадковими числами з нормального розподілу даних. Середнє значення першого масиву становитиме 5,0 зі стандартним відхиленням 1,0. Другий масив матиме середнє значення 10,0 зі стандартним відхиленням 2,0 (Рисунок 2.48):

```
import numpy as np
import matplotlib.pyplot as plt
x = np.random.normal(5.0, 1.0, 1000)
y = np.random.normal(10.0, 2.0, 1000)
```

```
plt.scatter(x, y)
plt.show()
```

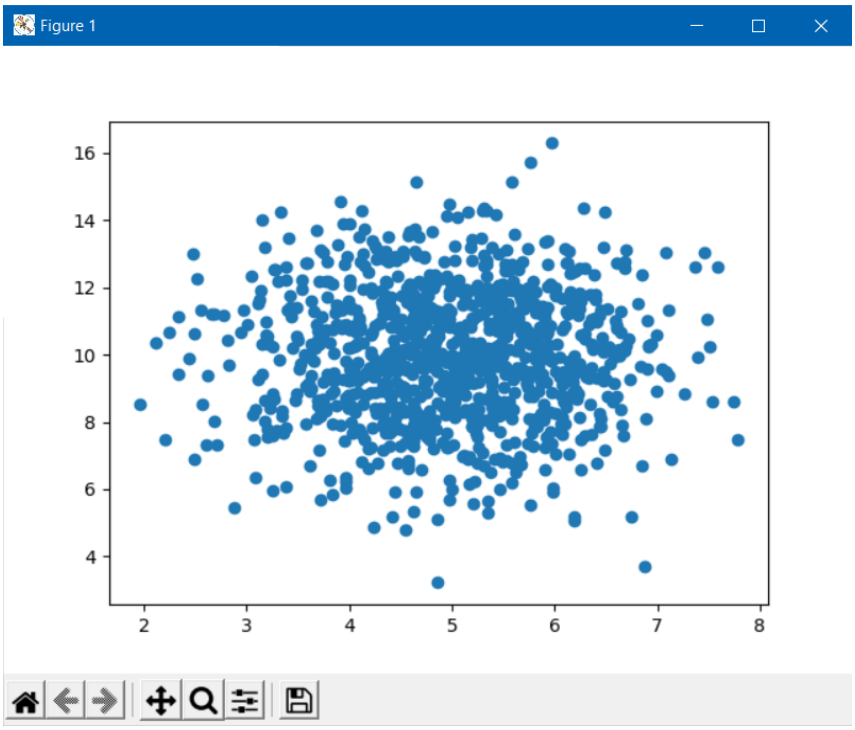


Рисунок 2.48 – Візуалізація даних в площині (x, y), розподілених за нормальним розподілом, які представляють результат роботи вищенаведеного коду.

Для створення великого набору даних для тестування з рівномірним розподілом даних використовується також функція uniform() модуля numpy.random, для рівномірного розподілу даних, uniform() дозволяє створення випадкових наборів даних в заданому діапазоні low (0.0) - high (5.0).

Приклад: Створимо масив, що містить 250 випадкових чисел з рухомою комою в діапазоні значень від 0 до 5:

```
# Метод uniform() модуля random пакету Numpy рівномірного
розподілу даних
import numpy
x = numpy.random.uniform(0.0, 5.0, 10)
print(f'random numbers from (випадкові числа від) 0 up to (до) 5 =
{x}')
```

Результат роботи коду:
random numbers from (випадкові числа від) 0 up to (до) 5 = [2.4818389 1.48367808 4.83914579 1.96379224 0.72367236 3.22207648 0.23342641 3.97202865 4.72824752 2.85487897]

Для візуалізації набору даних ми можемо побудувати гістограму зі згенерованими нами даними. Ми будемо використовувати метод uniform() модуля matplotlib для побудови гістограми (Рисунок 2.49):

```
# Гістограма (uniform - рівномірний)
import numpy as np
import matplotlib.pyplot as plt
x = np.random.uniform(0.0, 5.0, 250)
```

```
# low (0.0) - нижня межа вихідного інтервалу випадкової величини
# high (5) - верхня межа вихідного інтервалу випадкової величини
# при побудові гістограми похибку задати неможливо
plt.hist(x, 5)
plt.show()
```

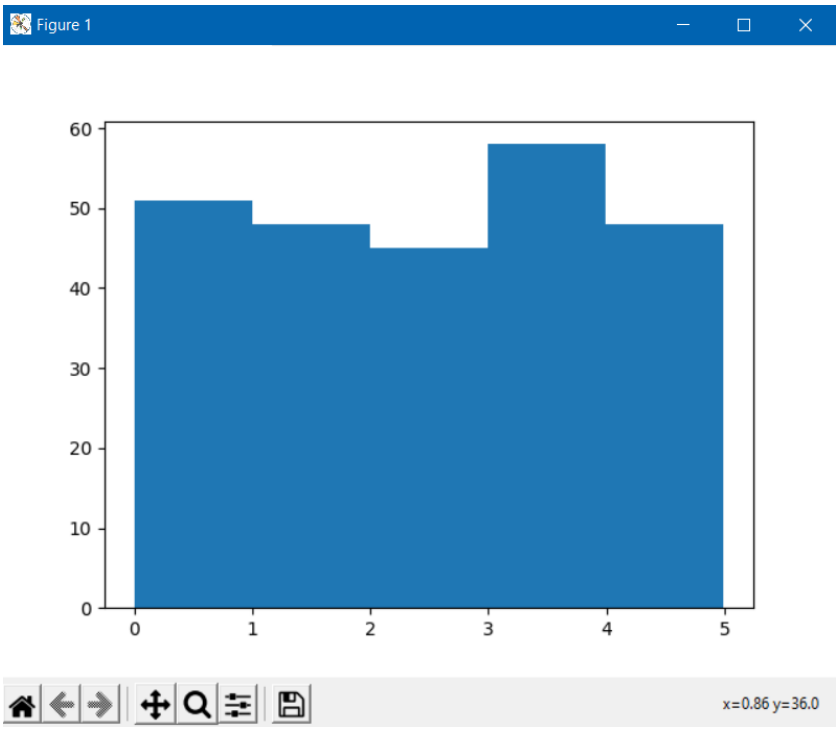


Рисунок 2.49 – Візуалізація результату роботи попереднього коду

Зімітуємо проведення експерименту, візуалізуємо дані, де кожне значення в наборі даних представлено крапкою, для подальшого їх аналізу, використаємо для цього метод scatter() пакету matplotlib, для цього задаємо два масиви однакової довжини, один для значень незалежної змінної x, а інший – для значень залежної змінної y.
x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11,12, 9, 6]
y = [99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86]
де
x – масив, в якому зберігаються дані про кількість пропущених лекцій студентами групи при вивченні конкретної дисципліни.
y – масив, в якому зберігаються дані про бали, набрані студентами групи на екзамені.

Метод scatter() використовується для складання діаграми розсіювання (англ. scatter – розсіювання). Використовуємо метод scatter() для візуалізації даних (Рисунок 2.50):

```
import matplotlib.pyplot as plt
x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
plt.scatter(x, y)
plt.show()
```

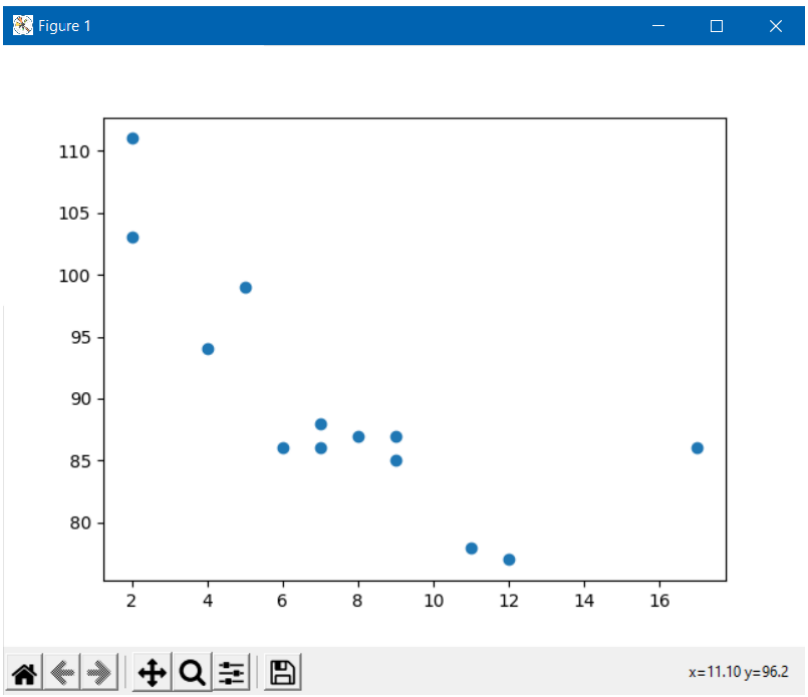


Рисунок 2.50 – Візуалізація результату роботи попереднього коду.

2.4.5. Лінійна регресія в python

Наука про дані та машинне навчання сприяють прогресу медицини, розпізнаванню зображень, розробці автономних транспортних засобів, рішенням у фінансовому та енергетичному секторах, розвитку соціальних мереж тощо. Важливою частиною цього є лінійна регресія. Лінійна регресія є одним із фундаментальних методів статистики та машинного навчання.

Лінійна регресія застосовується для аналізу експериментальних даних, в машинному навчанні, статистичному моделюванні тощо. Поняття регресія використовується, коли необхідно знайти зв'язок між змінними. У машинному навчанні, аналізі експериментальних даних, статистичному моделюванні тощо цей зв'язок використовується для прогнозування результатів майбутніх подій.

Лінійна регресія використовує відношення між точками даних для проведення прямої лінії через усі точки. Цю лінію можна використовувати для прогнозування майбутніх значень.

Проста лінійна регресія – це лінійний підхід для моделювання зв’язку між однією залежною та однією незалежною змінною для прогнозування результатів майбутніх подій.
У python є методи для знаходження зв’язку між точками даних та проведенням лінії лінійної регресії.

Багатофакторна або багатовимірна лінійна регресія – це лінійний підхід для моделювання зв’язку між однією залежною змінною та декількома незалежними змінними для прогнозування результатів майбутніх подій.

Проста (або одновимірна) лінійна регресія відповідає лінійній моделі y = k*x + b з коефіцієнтами k = slope, b = intercept, тут x – незалежна змінна, y – залежна змінна. При цьому мінімізується сума квадратів відстаней між спостережуваними даними для залежної змінної та передбаченим лінійним наближенням цих даних (на Рисунку 2.51 – це сума квадратів довжин червоних відрізків).

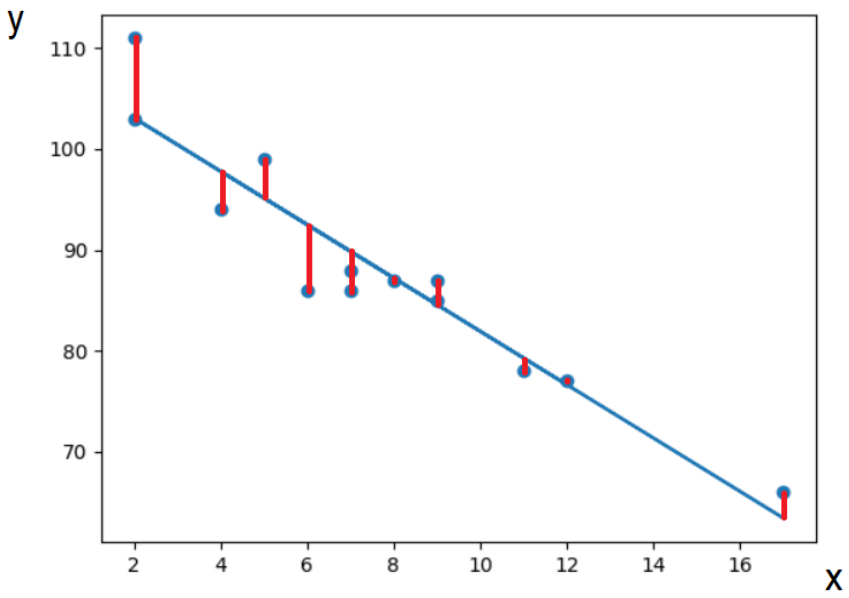


Рисунок 2.51 – Схематичне зображення довжин відрізків (червоний колір), сума квадратів довжин яких мінімізуєтеся в методі найменших квадратів. А саме, мінімізується сума квадратів відстаней між спостережуваними даними для залежної змінної та передбаченим лінійним наближенням цих даних (на рисунку – це сума квадратів довжин червоних відрізків).

Метод scipy.stats.linregress(x, y=None, alternative='two-sided‘) обчислює лінійну регресію за методом найменших квадратів для двох наборів вимірювань [25, 38]. Параметри методу:
x, y – масиви даних. Обидва масиви повинні мати однакову довжину. Якщо вказано лише x, а y=None, то це має бути двовимірний масив і метод linregress(x) буде мати вигляд: .linregress(x[0], x[1]).
Alternative = {‘two-sided’, ‘less’, ‘greater’} визначає альтернативні можливості методу (за замовчуванням — «*two-sided*»):
«*two-sided*»: нахил лінії регресії відмінний від нуля;
«*less*»: нахил лінії регресії менший за нуль;
«*greater*»: нахил лінії регресії більший за нуль.
Метод .linregress() повертає такі результати:
slope (нахил) – нахил лінії регресії.
intercept (перетин) – перетин осі ординат (y) лінією регресії.
r value (значення) – значення коефіцієнта кореляції.
p value – значення ймовірності того, що нахил лінії регресії дорівнює нулю.
stderr – середнє квадратичне відхилення (стандартна похибка) розрахунку нахилу лінії регресії.

Побудуємо графічне зображення лінійної регресії (Рисунок 2.52):

```
# Графічне зображення лінійної регресії
import matplotlib.pyplot as plt
from scipy import stats
x = [5, 7,  8, 7, 2, 17,  2,  9,  4, 11,  12, 9, 6]
y = [99,86,87,88,111,66,103,87,94,78,77,85,86]
# нахил лінії регресії;  перетин осі ординат лінією регресії в
точці
# x=0; коефіцієнт  кореляції; ймовірність того, що нахил лінії
```

```
# регресії = 0; стандартна похибка розрахунку нахилу лінії
регресії
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):
     return slope * x + intercept
# Замість використання циклу for, функція map() дає #можливість
застосувати функцію myfunc до кожного #елементу  ітерованого
об'єкту
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

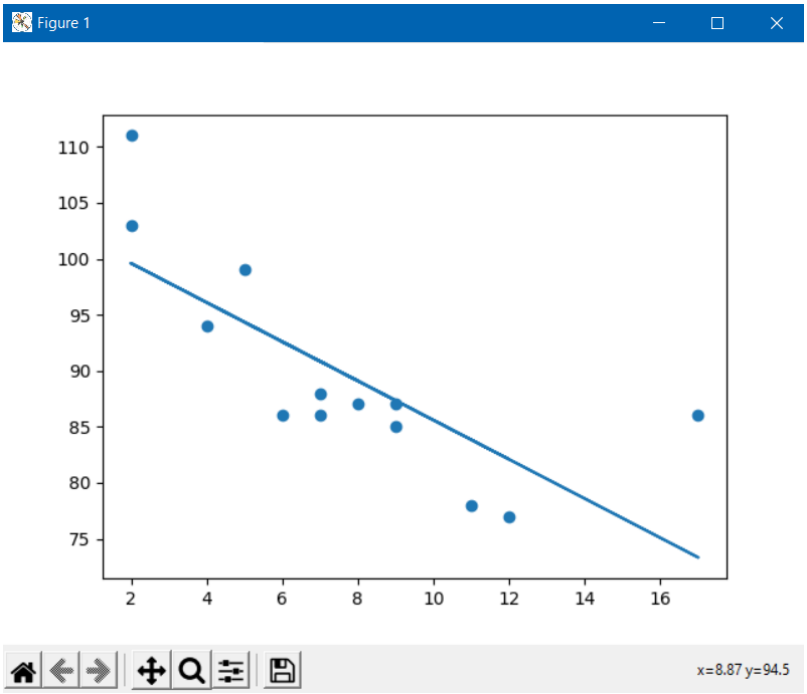


Рисунок 2.52 – Графічне зображення лінійної регресії – результат роботи попереднього коду.

Розглянемо графічну візуалізацію параметрів slope, intercept в методі scipy.stats.linregress(x, y=None, alternative='two-sided‘) лінійної регресії (Рисунок 2.53).
Для оцінки якості лінійної інтерполяції експериментальних даних розраховується такий показник як коефіцієнт кореляції. У математичній статистиці коефіцієнт кореляції – показник, що характеризує ступінь статистичного зв'язку між двома чи кількома випадковими величинами.

Якщо коефіцієнт кореляції описує зв'язок між двома випадковими величинами, він називається простим, якщо між однією випадкової величиною та його групою, то множинним.
Простий коефіцієнт кореляції (Пірсона) r обчислюється за такою формулою:

$$r = \frac{\sum(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum(x_i-\bar{x})^2 \sum(y_i-\bar{y})^2}},$$

де x, y – випадкові змінні. Значення коефіцієнта кореляції завжди знаходяться в діапазоні від -1 до 1.

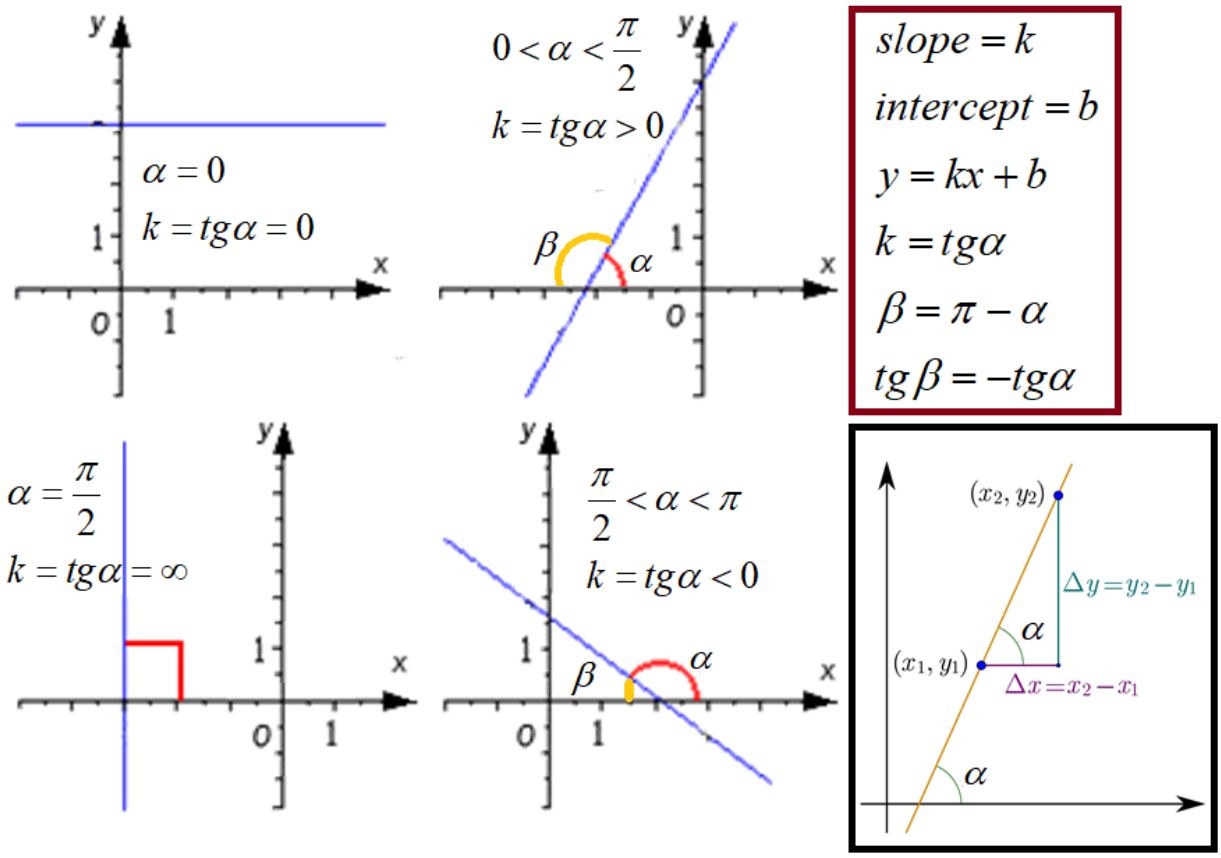


Рисунок 2.53 – Графічна візуалізація параметрів slope, intercept в методі scipy.stats.linregress(x, y=None, alternative='two-sided‘) лінійної регресії.

Коефіцієнт кореляції може застосовуватися з метою оцінки значущості регресійних моделей. Коефіцієнт кореляції широко застосовується в аналізі даних для відбору змінних в аналітичних моделях та виявлення найбільш значущих ознак з точки зору задачі, яка розв'язується. Значення коефіцієнта кореляції завжди знаходяться в діапазоні від -1 до 1 та інтерпретуються таким чином (Таблиця 2.5):

- якщо коефіцієнт кореляції близький до 1, між змінними спостерігається позитивна кореляція. Іншими словами, відзначається високий рівень зв'язку між змінними. В даному випадку, якщо значення змінної *x* зростатимуть, то й вихідна змінна *y* також збільшуватиметься;
- якщо коефіцієнт кореляції близький до -1 це означає, що між змінними має місце сильна негативна кореляція. Іншими словами, поведінка вихідної змінної *y* буде протилежною поведінки вхідної *x*. Якщо значення *x* зростатиме, то *y* зменшуватиметься, і навпаки;
- проміжні значення, близькі до 0, будуть вказувати на відсутність кореляції між змінними. Іншими словами, поведінка змінної *x* не буде впливати на поведінку *y* (і навпаки).

Обчислення коефіцієнта кореляції r для оцінки взаємозв'язку між даними здійснюється за допомогою модуля scipy.stats [25].

Важливо знати, яким є зв’язок між значеннями незалежної змінної x та значеннями залежної змінної y, якщо немає зв’язку, то лінійну регресію неможливо використати, щоб щось передбачити. За допомогою модуля scipy.stats обчислюється коефіцієнт кореляції, для цього необхідно передати методу .linregress() значення x та y. Розрахувавши лінійну

регресію, ми можемо використовувати зібрану інформацію для прогнозування невідомих даних.

Приклад: Спробуємо передбачити бали на екзамені для студента, який пропустив 10 занять протягом семестру.

Таблиця 2.5. Інтерпретація коефіцієнта кореляції.

Значення r	Рівень зв'язку між змінними
0,75 – 1.00	дуже високий позитивний
0,50 – 0.74	високий позитивний
0,25 – 0.49	середній позитивний
0,00 – 0.24	слабкий позитивний
0,00 – -0.24	слабкий негативний
-0,25 – -0.49	середній негативний
-0,50 – -0.74	високий негативний
-0,75 – -1.00	дуже високий негативний

Наступний приклад ілюструє передбачення балів на екзамені для студента, який пропустив N занять протягом семестру:

```
# Передбачення балів на екзамені для студента, який пропустив 10
занять протягом семестру
from scipy import stats
x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]
# x – кількість пропущених лекцій
y = [99, 86, 87, 88, 100, 40, 100, 87, 94, 78, 70, 85, 86]
# y – бали на екзамені
slope, intercept, r, p, std_err  = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

score = myfunc(10)
print(f'exam score for student missing 10 lectures is {score}')
print(f'коефіцієнт кореляції =  {r}')
Результат роботи коду:
exam score for student missing 10 lectures is  85.59308314937454
коефіцієнт кореляції =  -0.940171761326698
```

Багатовимірна (або багатофакторна) регресія схожа на одновимірну лінійну регресію, але з більш ніж однією незалежною змінною, тобто ми намагаємось передбачити значення на основі двох або більшої кількості незалежних змінних. Наприклад, у випадку однієї залежної величини (y = Size) і 2-ох незалежних X = (x_1 = Time, x_2 = Path) ми будемо наближати залежність y = y(x_1, x_2) площиною у тривимірному просторі. Маємо набір даних (файл _56day.txt), який містить інформацію про розмір біооб'єкту, час його руху під впливом зовнішньої сили та пройдений шлях.

Ми можемо передбачити розміри біооб'єкта, якщо задано час його руху, але з множинною регресією ми можемо ввести більше змінних, таких як пройдений шлях, в'язкість рідини тощо, щоб зробити прогноз більш точним (Таблиця 2.6).

Таблиця 2.6. Приклад експериментальних даних для випадку однієї залежної величини (y = Size) і 2-ох незалежних X = (x_1 = Time, x_2 = Path)

Size	Time	Path
160	5.29	1300
200	9.77	1800
180	9.94	1600
300	12.82	1100
200	6.15	1300
200	11.31	2000
180	10.39	1700
250	6.54	1000
400	13.52	1200
350	7.26	800

Задачі багатовимірної лінійної регресії розв'язуються з застосуванням класу LinearRegression() модуля linear_model пакету sklearn.

sklearn.linear_model.LinearRegression (fit_intercept=True, copy_X=True):
fit_intercept: [boolean, за замовчуванням – True], чи обчислювати intercept моделі;
copy_X: [boolean, значення за замовчуванням – True], якщо істинно, то робиться копія X.
Множинна лінійна регресія відповідає лінійній моделі з коефіцієнтами:
w = (w_0, …, w_p), щоб мінімізувати залишкову суму квадратів між спостережуваними даними і передбаченим лінійним наближенням цих даних.

$$y = w_0 + w_1 x_1 + w_2 x_2$$

y – "Size", x = (x_1, x_2), де x_1 – "Time", x_2 – "Path"

Наведемо приклад двох незалежних змінних x = (x_1, x_2) і однієї залежної y, при цьому багатовимірна лінійна регресія відповідає лінійній моделі $y = w_0 + w_1 x_1 + w_2 x_2$ з коефіцієнтами w = (w_0, w_1, w_2), щоб мінімізувати залишкову суму квадратів між спостережуваними даними і передбаченим лінійним наближенням цих даних (на рисунку – це сума квадратів довжин жовтих відрізків, проведених з експериментальних точок (червоний колір) до площини, на якій знаходяться передбачені значення залежної змінної) (Рисунок 2.54).

Метод fit() пакету sklearn приймає масиви x, y і повертає коефіцієнти w=(w0,…wp) багатовимірної лінійної регресії.

Метод predict() пакету sklearn в якості параметрів приймає x=(x_1 ….x_p) із застосуванням переданих методом fit() коефіцієнтів w багатовимірної лінійної регресії повертає прогнозоване значення y.

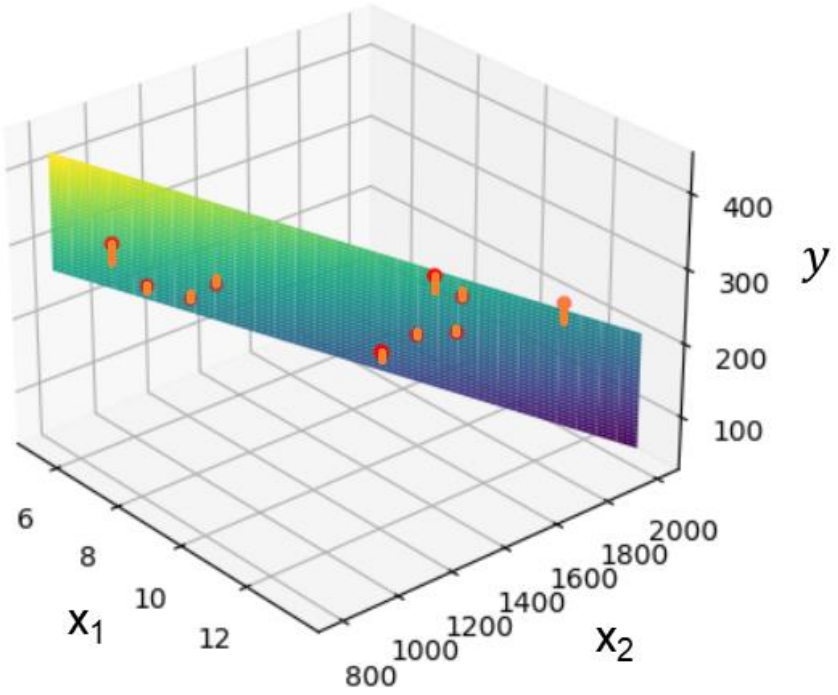


Рисунок 2.54 – Графічна ілюстрація мінімізації залишкової суми квадратів між спостережуваними даними і передбаченим лінійним наближенням цих даних (на рисунку – це сума квадратів довжин жовтих відрізків, проведених з експериментальних точок (червоний колір) до площини, на якій знаходяться передбачені значення залежної змінної)

Метод genfromtxt (*fname*, *delimiter=None*, *skip_header=0, ……*) пакету numpy завантажує дані з текстового файлу, *fname* - файл, назва файлу, список або генератор для читання. Кожен рядок після перших рядків skip_header розділяється за символом роздільника *delimiter*, параметр skip_header=1 – кількість рядків, які потрібно пропустити на початку файлу. Наприклад:

```
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
# зчитуємо дані з файлу в масив numpy, параметр skip_header=1 -
кількість рядків, які
# потрібно пропустити на початку файлу, delimiter - рядок для
значення роздільника.
df = np.genfromtxt("TestTxtFiles/_56 day.txt", delimiter=",",
skip_header=1)
X1 = df[:, 1]                # 1-ий стовпчик, будь-який рядок
X2 = df[:, 2]
#транспонування матриці обчислюється шляхом заміни стовпців на
рядки в матриці
X = np.array([X1, X2]).transpose()
y = df[:, 0]
regr = linear_model.LinearRegression(fit_intercept=True)
# метод .fit() повертає коефіцієнти регресії w=(w0,…wp)
багатовимірної лінійної регресії
regr.fit(X, y)
```

```
# Розрахунок значень коефіцієнтів регресії
print(f'regr.coef = {regr.coef_}')                    # coef_ -
атрибут об'єкту regr - коефіцієнти регресії
print(f'regr.intercept= {regr.intercept_}')
# 3D площина зображує результат багатовимірної лінійної регресії
number = 30
x1 = np.linspace(min(X1), max(X1), number)
x2 = np.linspace(min(X2), max(X2), number)
y_model = np.zeros((number, number))
for i in range(number):
    for j in range(number):
        y_model[i, j] = regr.predict([[x1[i], x2[j]]])
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(x1, x2, y_model, cmap='viridis', edgecolor='none')
ax.set_title('Size(Time, Path)=y(X1, X2)')

ax.scatter(X1, X2, y, marker='o', color='red')
ax.set_xlabel('Time')
ax.set_ylabel('Path')
ax.set_zlabel('Size')
plt.show()
# Передбачаємо розміри біооб'єкта якщо час = 6 с, шлях = 1000 мкм
predicted_size = regr.predict([[6, 1000]])
print(f'predicted_size = {predicted_size}')
Результат роботи коду (Рисунок 2.55):
regr.coef = [19.12652378 -0.17845126]
regr.intercept= 310.40519649812984
predicted_size = [246.71307749]
```

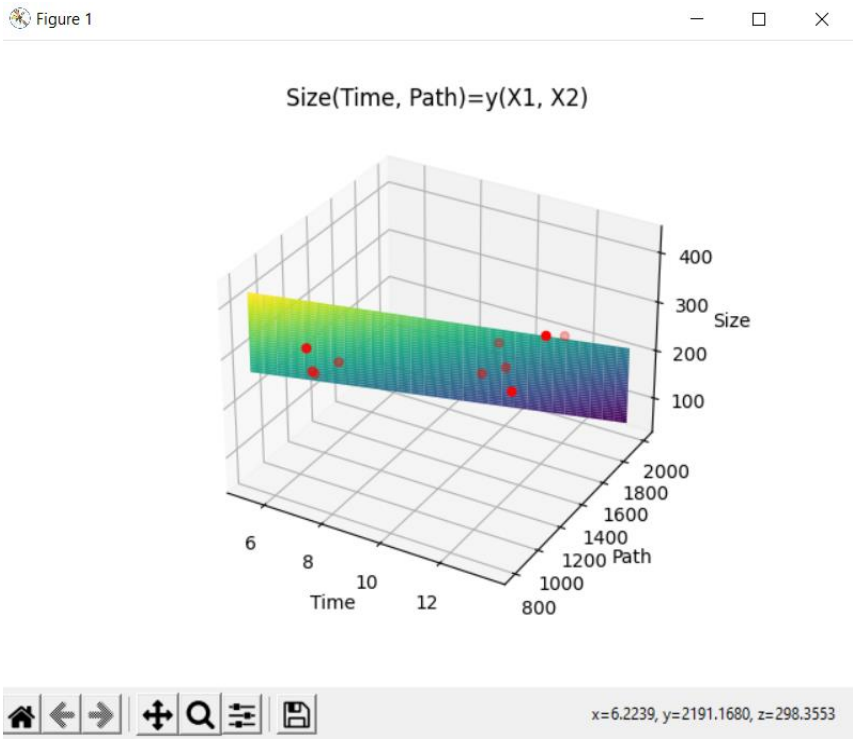


Рисунок 2.55 – Графічна візуалізація результату роботи попереднього коду

В попередньому прикладі отримано коефіцієнти регресії w_p . Коефіцієнти регресії – це коефіцієнти, які описують зв'язок з залежної змінної з незалежними змінними. Приклад: якщо x_1 – незалежна змінна, а коефіцієнт $w_1 = 2$, тоді $y = 2x_1$, якщо не змінювати всі інші залежні змінні окрім x_1. У цьому випадку ми можемо розрахувати значення коефіцієнта розміру біооб'єкта в залежності від шляху та розрахувати значення коефіцієнта розміру біооб'єкта від часу руху. Результати, які ми отримаємо, покажуть, що буде, якщо ми збільшимо або зменшимо одну з незалежних змінних: шлях або час руху.

2.4.6. Особливості масштабування даних в python

Якщо Ваші дані мають різні значення і навіть різні одиниці вимірювання, їх може бути важко або неможливо порівняти. Що таке секунди в порівнянні з мікронами? Відповідь на цю проблему – це масштабування. Ми можемо масштабувати дані, поділивши їх на деякі обрані нами значення, в результаті ми отримуємо дані, які легше порівняти. Існують різні методи масштабування даних, у цій лекції ми будемо використовувати метод під назвою стандартизація.

Метод стандартизації використовує таку формулу для всіх i – номерів значень в наборі даних:

$$z_i = \frac{x_i - \bar{x}}{\sigma},$$

де z_i -значення в масштабованому наборі даних, x_i - значення початкового набору даних, $\bar{x}$ - середнє значення і σ - стандартне відхилення (іншими словами середнє квадратичне відхилення).

Сформуємо той самий набір даних, який ми використовували у попередньому прикладі множинної регресії, але цього разу дані будуть масштабовані:

```
# Масштабування даних
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()

df = pandas.read_csv('TestTxtFiles/_56 day.txt')
X = df[['Time', 'Path']]
scaledX = scale.fit_transform(X)
print(f'scaledX = {scaledX}')

Результат роботи коду:
scaledX =
[[-1.47534062 -0.22052714]
 [ 0.17333136  1.15776748]
 [ 0.23589258  0.60644963]
 [ 1.29575314 -0.77184498]
 [-1.15885448 -0.22052714]
 [ 0.74006236  1.70908532]
 [ 0.40149579  0.88210855]
 [-1.0153317  -1.04750391]
 [ 1.55335813 -0.49618606]
 [-0.75036656 -1.59882175]]
```

Передбачаємо розміри біооб’єкта, якщо час = 6 с, шлях = 1000 мкм з масштабуванням даних:

```
# Передбачаємо розміри біооб’єкта якщо
# час = 6 с, шлях = 1000 мкм з масштабуванням даних
```

```
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
df = pandas.read_csv('TestTxtFiles/_56 day.txt')
X = df[['Time', 'Path']]
y = df['Size']
scaledX = scale.fit_transform(X)
regr = linear_model.LinearRegression()
regr.fit(scaledX, y)
scaled = scale.transform([[6, 1000]])
predicted_size = regr.predict([scaled[0]])
print(f'predicted_size = {predicted_size}')

Результат роботи коду:
predicted_size = [246.71307749]
```

2.4.7. Поліноміальна регресія в python

Якщо Ваші точки даних не відповідають лінійній регресії (пряма лінія через усі точки даних), то для цих даних можна застосувати поліноміальну регресію. Багаточленом або поліномом однієї змінної в математиці називається вираз такого виду:

$$y = c_0 + c_1 x + \cdots + c_n x^n,$$

де c_0 – intercept, перетин осі ординат лінією регресії, якщо немає інших коефіцієнтів, c_1 – slope, нахил лінії регресії для прямої, якщо немає інших коефіцієнтів.

Якщо є інші відмінні від нуля коефіцієнти, то c_0 і c_1 – це просто коефіцієнти поліноміальної регресії.

Для розв'язання задач поліноміальної регресії використовується клас poly1d пакету numpy [38].

Клас numpy.poly1d(*c_or_r, r = False, variable = None*) будує поліном з використанням коефіцієнтів поліному p (метод.polyfit (*x, y, deg*)). Параметри класу numpy.poly1d():

c_or_r (coefficient or root) – це масив (або список) коефіцієнтів полінома, які стоять множниками при степенях змінної, що зменшуються, або, якщо значення другого параметра True, то це корені полінома (значення, де поліном дорівнює 0).

poly1d([1, 2, 3]) => $x^2 + 2x + 3$ – коефіцієнти полінома

poly1d([1, 2, 3], True) => $(x-1)(x-2)(x-3) = x^3 - 6x^2 + 11x - 6$ – корені полінома

r – булева змінна, якщо r = True, то перший параметр c_or_r вказує корені полінома; за замовчуванням встановлено значення r = False.

variable str – змінює назву змінної, яка використовується під час друку поліному з x на інше значення (variable), наприклад z:

```
p = np.poly1d([1, 2, 3], variable ='z')
print(p)
```

друкується: $z^2 + 2z + 3$

Метод .polyfit(*x, y, deg*) пакету numpy – повертає коефіцієнти полінома p.

Параметри:

x, y – масиви координат точок.

deg(int) – степінь полінома, який використовується для поліноміальної регресії.

Метод .polyfit() повертає коефіцієнти полінома:
p (c_n, c_{n-1}, … , c_1, c_0) – коефіцієнти полінома, найвища степінь вказується першою.

Метод .linspace() рівномірно розташовує числа на вказаному інтервалі. Наприклад, .linspace(1, 22, 100) повертає 100 точок рівномірно розподілених в діапазоні від 1 до 22.

Поліноміальна регресія, як і лінійна регресія, використовує співвідношення між змінними x та y, щоб знайти найкращий спосіб провести криву через точки даних. Наприклад:

```
import numpy
import matplotlib.pyplot as plt
x = [1, 2, 3, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 18, 19, 21,
22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
mymodel = numpy.poly1d( numpy.polyfit (x, y, 3))
myline = numpy.linspace (1, 22, 100)
plt.scatter(x, y)
plt.plot( myline, mymodel( myline))
plt.show()
```

Методу polyfit() передаємо в якості параметрів дані x, y та спепінь полінома = 3, а повертає p - коефіцієнти полінома і передає класу poly1d. Створюємо екземпляр mymodel класу poly1d, в якості атрибутів якого передаємо результат роботи методу polyfit (Рисунок 2.56).

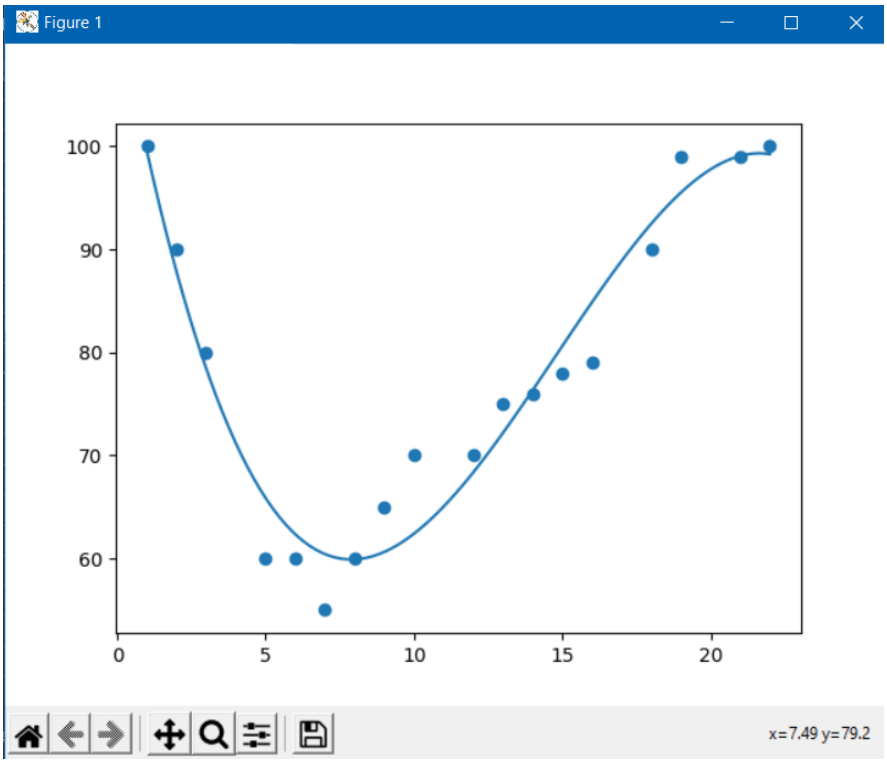


Рисунок 2.56 – Графічна ілюстрація результату роботи попереднього коду.

2.4.8. R^2 (коефіцієнт детермінації). Модуль sklearn.metrics

Важливо знати, наскільки добре співвідношення між значеннями x та y, і якщо між ними немає зв'язку, то поліноміальну регресію неможливо використати, щоб щось передбачити.

Взаємозв'язок між наборами даних вимірюється на основі значень параметру, який називається коефіцієнтом детермінації (r^2). Значення коефіцієнта детермінації коливається від 0 до 1, де 0 означає відсутність зв'язку, а 1 означає 100% зв'язок. Коефіцієнт детермінації показує, яка частина коливань результативної ознаки (y) зумовлена коливанням факторної ознаки (x). Наприклад, якщо $r^2 = 0,76$, то на 76% зміна y залежить від зміни x, а на $(1-r^2) = 0,24$, тобто на 24%, від інших, нерозглянутих факторів.

Python і модуль sklearn.metrics та метод r2_score() обчислює це значення, для чого потрібно передати дані – масиви x, y та спепінь полінома.

Для розрахунку R-квадрат (коефіцієнта детермінації) розраховуються такі величини (Рисунок 2.57):

- Середнє значення y:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i .$$

- Сума квадратів залишків, також називається залишковою сумою квадратів (f – це набір прогнозованих значень, точки на кривій):

$$S_{res} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2 .$$

- Загальна сума квадратів (є пропорційною дисперсії даних):

$$S_{tot} = \sum_i (y_i - \bar{y})^2 .$$

Найбільш загальним визначенням коефіцієнта детермінації (R^2) є:

$$R^2 = 1 - \frac{S_{res}}{S_{tot}} .$$

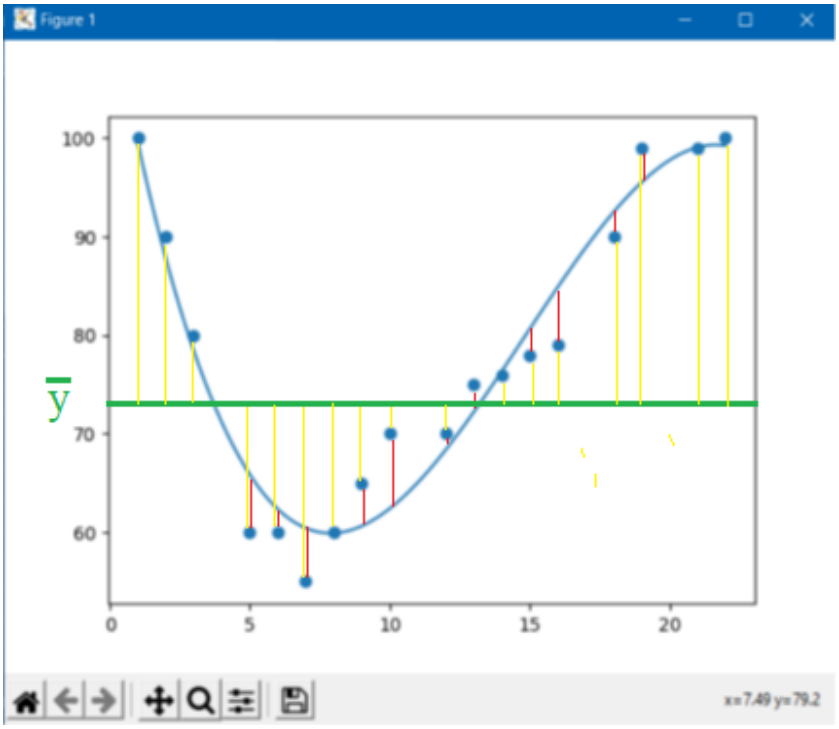


Рисунок 2.57 – Графічна ілюстрація мінімізації суми квадратів залишків, яка також називається залишковою сумою квадратів (сума квадратів довжин червоних відрізків на рисунку).

Наведемо приклад коду для оцінювання наскільки добре дані описуються поліноміальною регресією:

```
# Оцінювання наскільки добре дані описуються поліноміальною
регресією
import numpy
from sklearn.metrics import r2_score
```

```
x = [1, 2, 3, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 18, 19, 21,
22]
y = [100, 90, 80, 60, 60, 55, 60, 65, 70, 70, 75, 76, 78, 79, 90,
99, 99, 100]
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
print(f'r2 = {r2_score(y, mymodel(x))}')
Результат роботи коду:   r² = 0.9432150416451026
```

f - це набір прогнозованих значень, f = mymodel(x).

Методу r2_score() в якості параметрів передаємо експериментальні значення y та прогнозовані значення mymodel(x), тобто метод r2_score() розраховує R^2.

Наведемо також приклад, коли поліноміальна регресія не підходить для прогнозування (Рисунок 2.58):

```
# Приклад, коли поліноміальна регресія не підходить для
прогнозування
import numpy
import matplotlib.pyplot as plt
x = [89, 43, 36, 36, 95, 10, 66, 34, 38, 20, 26, 29, 48, 64, 6,5,
36, 66, 72, 40]
y = [21, 46, 3, 35, 67, 95, 53, 72, 58, 10, 26, 34, 90, 33, 38,
20, 56, 2, 47, 15]
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
myline = numpy.linspace(2, 95, 100)
plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```

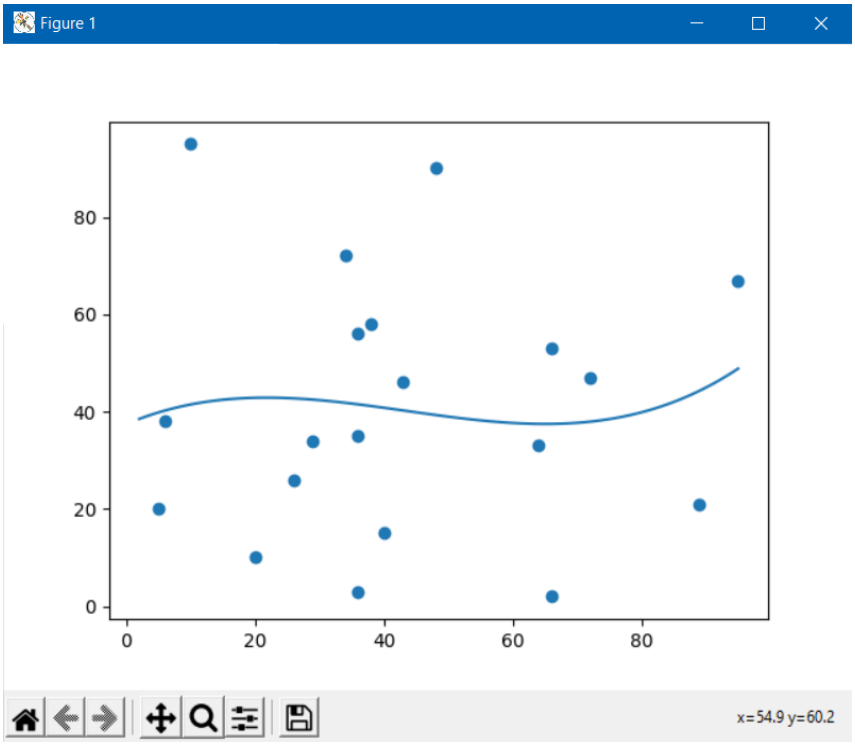


Рисунок 2.58 – Графічна ілюстрація результату роботи попереднього коду для випадку, коли поліноміальна регресія не підходить для прогнозування.

Нижче наведений код ілюструє, що значення r-квадрата для попереднього прикладу є дуже низьким:

```
import numpy
from sklearn.metrics import r2_score

x = [89, 43, 36, 36, 95, 10, 66, 34, 38, 20, 26, 29, 48, 64, 6,5,
36, 66, 72, 40]
y = [21, 46, 3, 35, 67, 95, 53, 72, 58, 10, 26, 34, 90, 33, 38,
20, 56, 2, 47, 15]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
print(f'r2 = {r2_score(y, mymodel(x))}')

Результат роботи коду:
r2 = 0.009952707566680652
```

2.4.9. Побудова моделі: поділ даних на тренувальні/тестувальні, оцінка якості моделі, передбачення з використанням навченої моделі

Почніть з набору даних, який потрібно перевірити. Наш набір даних ілюструє 100 студентів та їх ставлення до навчання в університеті.

- Дані по осі x означають кількість годин, які витрачає студент на підготовку до тестування.
- Дані по осі y відображають кількість неправильних відповідей на запитання тесту.

Нижченаведений код створює та візуалізує графічно навчальний комплект даних (Рисунок 2.59):

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)                                    # seed –
насіння, сім'я
# середній час підготовки студента до тесту 3 години
x = numpy.random.normal (3, 1, 100)
# В середньому 150 неправильних відповідей
# при одній годині підготовки до всіх лекцій
y = numpy.random.normal (150, 40, 100) / x
plt.scatter(x, y)
mymodel = numpy.poly1d (numpy.polyfit (x, y, 4))
myline = numpy.linspace(0, 6, 100)
plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Функція numpy.random.seed() використовується для встановлення початкового числа для алгоритму генератора випадкових чисел python. Ідея полягає в тому, що ми завжди отримуватимемо один і той же набір випадкових чисел для одного і того ж початкового числа на будь-якому комп'ютері при використанні методу seed(2).

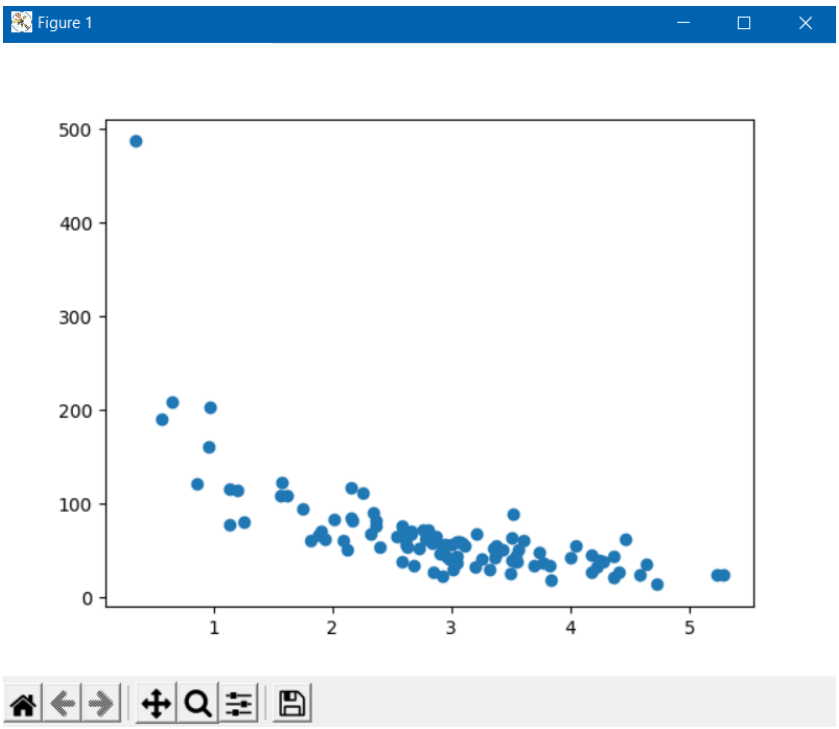


Рисунок 2.59 – Графічна ілюстрація результату роботи попереднього коду.

По осі абсцис – середній час підготовки студента до екзамену, годин.

По осі ординат – кількість неправильних відповідей на запитання тесту на екзамені.

У машинному навчанні ми створюємо моделі для прогнозування результатів певних подій. Щоб визначити, чи модель достатньо ефективно працює, необхідно використовувати метод під назвою Тренування/Тестування.

Що таке тренування/тестування? Це метод вимірювання точності Вашої моделі. Він називається тренування/тестування, тому що необхідно розділити набір даних на два набори: навчальний набір та набір тестування.

- Навчальний комплект повинен бути випадковим вибором 80% від вихідних даних, набір даних для тестування повинен бути рештою 20% підготовлених даних, тобто 80% для навчання та 20% для тестування. Це відсоткове співвідношення є оптимальним для більшості задач, оскільки більший тестовий набір може покращити оцінку загальної ефективності моделі, але може призвести до перенавчання на навчальних даних. Навпаки, менший тестовий набір може забезпечити недостатньо точну оцінку моделі.
- Ви тренуєте модель за допомогою навчального набору.
- Ви перевіряєте модель за допомогою набору тестів.

Навчити модель означає створити модель. Перевірити модель означає перевірити точність моделі.

Поділ даних на тренувальні/тестувальні відбувається таким чином:

- Навчальний комплект повинен бути випадковим вибором 80% від вихідних даних.
- Тестувальний комплект повинен бути рештою 20% підготовлених даних.

Нижченаведений код створює та візуалізує набір даних для тренування моделі (Рисунок 2.60):

```
# набір даних для тренування моделі
train_x = x[:80]
train_y = y[:80]
plt.scatter( train_x, train_y )
plt.show()
# набір даних для тестування моделі
```

```
test_x = x[80:]
test_y = y[80:]
plt.scatter(test_x, test_y)
plt.show()
```

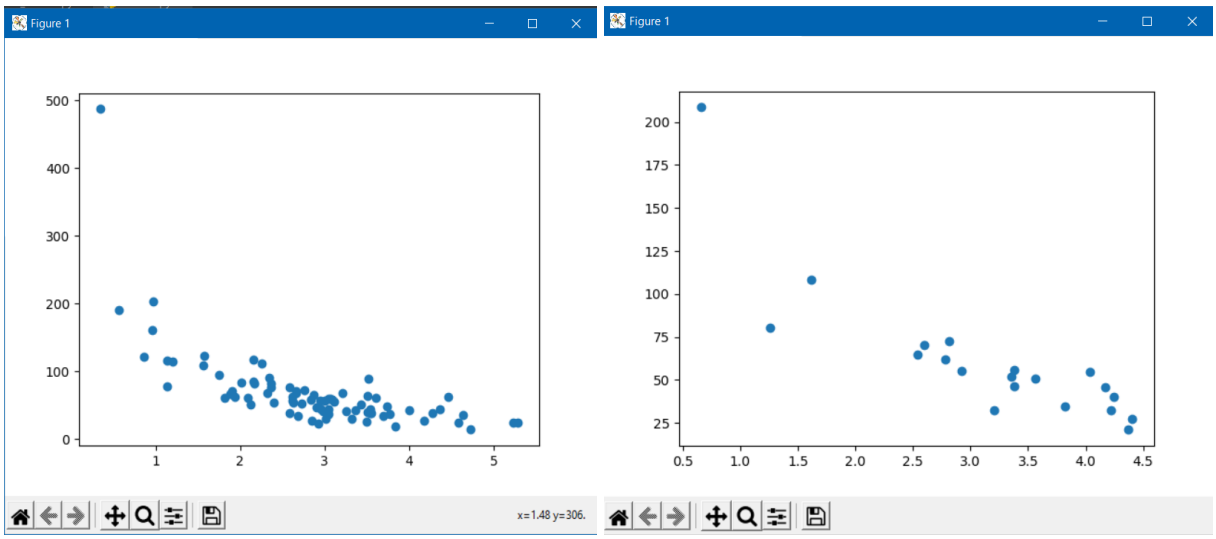


Рисунок 2.60 – Графічна ілюстрація результату роботи попереднього коду.

Здійснимо поліноміальну регресію для тренувальних даних (Рисунок 2.61):

```
import numpy
import matplotlib.pyplot as plt

numpy.random.seed(2)
x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x
train_x = x[:80]
train_y = y[:80]
test_x = x[80:]
test_y = y[80:]
mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))
myline = numpy.linspace(0, 6, 100)
plt.scatter(train_x, train_y)
plt.plot(myline, mymodel(myline))
plt.show()

import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)
x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x
train_x = x[:80]
train_y = y[:80]
test_x = x[80:]
test_y = y[80:]
mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))
r2 = r2_score(train_y, mymodel(train_x))
print(f'r2 = {round(r2, 3)}')
```

Ми побудували модель на тренувальному (навчальному) наборі даних, а перевіряємо якість моделі на тестовому наборі даних, підраховуючи коефіцієнт детермінації:

```
r2 = r2_score(test_y, mymodel(test_x))
print(f'r2 для тестувальної моделі = {round(r2, 3)}')
```

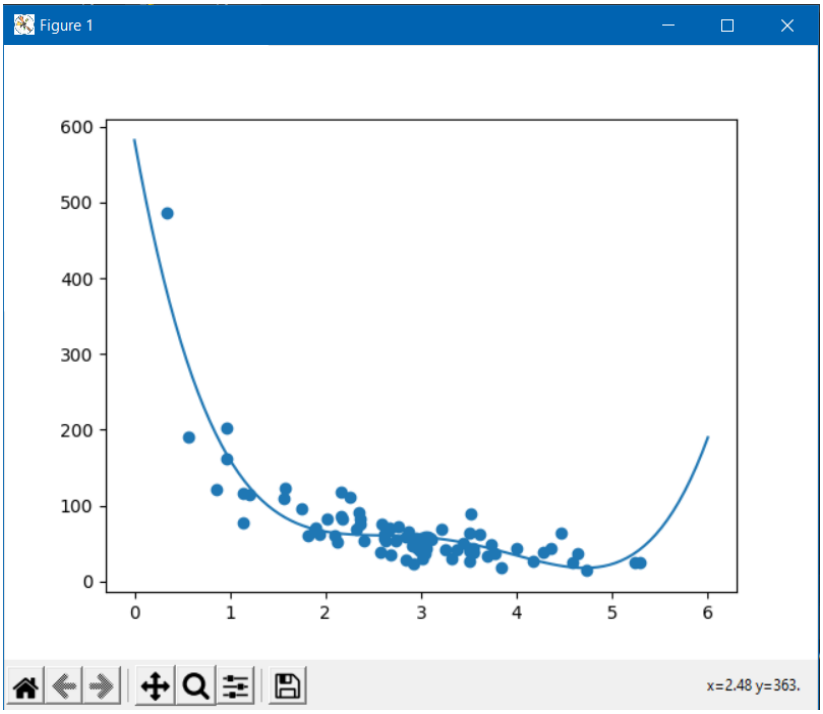


Рисунок 2.61 – Графічна ілюстрація результату роботи попереднього коду.

З'ясуємо, наскільки добре дані навчання вписуються в поліноміальну регресію:

Ми створили модель, для якої коефіцієнт детермінації 0.799 для навчального набору даних. При цьому ми побудували модель на тренувальному (навчальному) наборі даних, а перевірили на тестовому наборі даних.

```
Результат роботи коду:
r2 для тренувальної моделі = 0.799
r2 для тестувальної моделі = 0.809
```

Тепер, коли ми встановили, що наша модель працює коректно, ми можемо почати передбачати нові значення залежної змінної.

Приклад: Скільки неправильних відповідей буде в результатах здачі тесту студентом, якщо студент витратив 5 годин на підготовку?

```
print(f'n = {n} годин на підготовку, неправильних відповіде
mymodel(n) = {round(mymodel(n))}'
Результат роботи коду:
n = 5 годин на підготовку, неправильних відповіде mymodel(n) = 23
```

На основі моделі прогнозується, що студент дасть 23 неправильні відповіді (з 500 можливих відповідей) за умови 5 годин на підготовку до кожної лекції.

2.4.10. Приклад задачі апроксимації експериментальних даних

Наведемо наочний приклад підбору виду формули $\varphi(x)$ на основі літературних даних для задачі побудови аналітичної моделі і прогнозування прискорення поділу клітин в магнітному полі спеціальної просторової конфігурації [39].

Для прикладу розглянемо експериментальні дані з роботи [39], які описують вплив додавання хелатів заліза до середовища культивування та магнітного поля на ріст *E. coli* Nissle 1917 (EcN), при цьому культивування проводили протягом 28 годин з періодичним вимірюванням кількості живих клітин шляхом підрахунку в пристрої з лічильною камерою, яка зазвичай використовується для підрахунку клітин крові.

Криві росту *E. coli* Nissle 1917, культивованих за різних умов, зображені на 2.62 На Рисунок 2.62 використовуються позначення: N – кількість клітинних кластерів на мл клітинної суспензії, ST – стандартне середовище (контроль), ST+Fe - стандартне середовище з хелатами заліза без прикладення зовнішнього магнітного поля, ST+MF – стандартне середовище у зовнішньому 0,15 Тл, ST+MF+Fe - стандартне середовище з хелатами заліза в зовнішньому 0,15Тл.

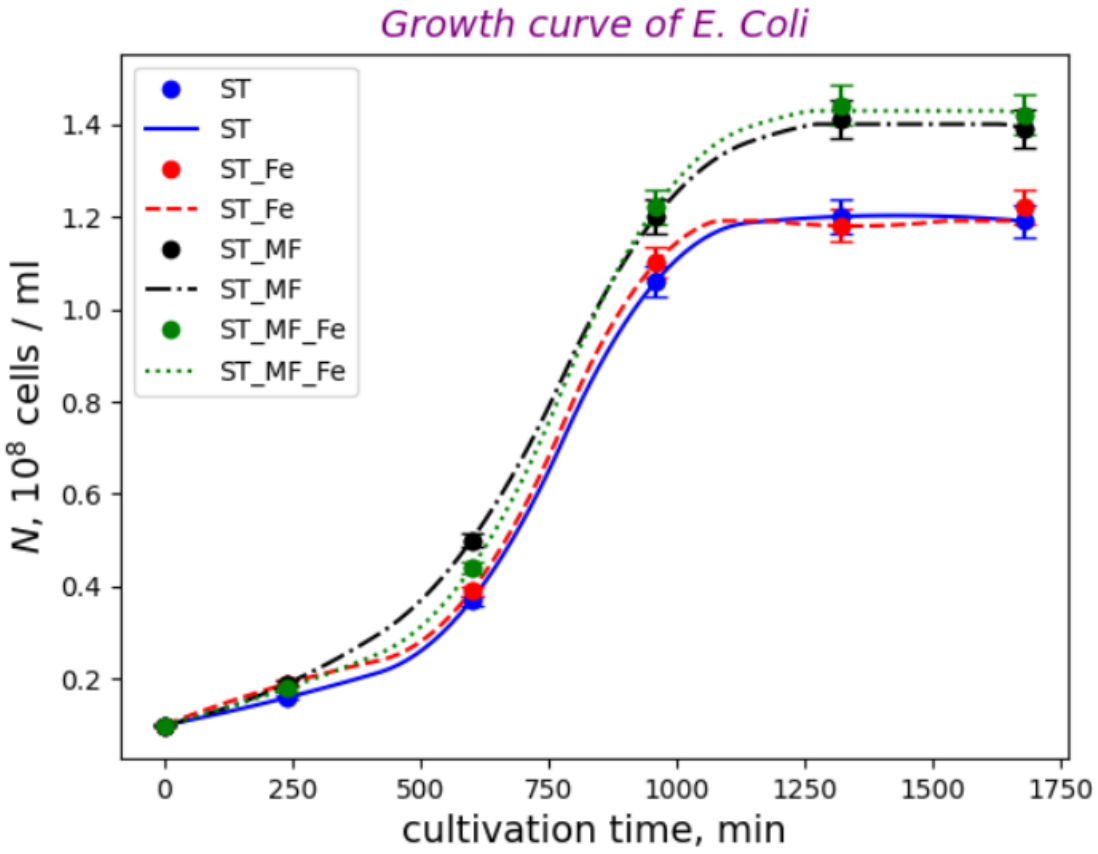


Рисунок 2.62 – Криві росту *E. coli* Nissle 1917, культивованих за різних умов: *N* – кількість клітинних кластерів на мл клітинної суспензії. N – кількість клітинних кластерів на мл клітинної суспензії, ST – стандартне середовище (контроль), ST+Fe - стандартне середовище з хелатами заліза без прикладення зовнішнього магнітного поля, ST+MF – стандартне середовище у зовнішньому 0,15 Тл, ST+MF+Fe - стандартне середовище з хелатами заліза в зовнішньому 0,15Тл.

Код на мові python, який здійснює нижченаведені розрахунки з розв’язання задачі апроксимації експериментальних даних заданою теоретичною функцією та їх графічну візуалізацію міститься у додатку до даного підручника.

На початковому етапі росту культури мікроорганізмів часова залежність чисельності бактерій підкорюється експоненціальному закону

$$N(t) = N_0 e^{\frac{t\,ln2}{\tau}}.$$

де N(t) і N_0 – концентрації бактерій у моменти часу t =0 і t , бактерій, а τ– час подвоєння. Для кожної кривої, показаної на Рисунок 2.62, час подвоєння кількості клітин можна визначити за кривими 1-4, а саме (див. тангенс кута нахилу прямої на Рисунок 2.63 та криві 1-4 на Рисунок 2.64)

$$\tau = \frac{t\,ln\,2}{ln\frac{N(t)}{N_0}};$$

τ_0 =276 хв для EcN, культивованих на стандартному середовищі (контроль), τ_{Fe} = 271 хв для EcN, культивованих на середовищі з додаванням хелатів заліза, τ_B =261 хв для EcN, культивованих на стандартному середовищі за наявності магнітного поля, та τ_{BFe} = 260 хв для EcN, культивованих на середовищі з додаванням хелату заліза в магнітному полі.

Знаючи час подвоєння кількості клітин EcN, культивованих в різних умовах росту, можна обчислити швидкість росту, використовуючи таку залежність

r =ln2/τ.

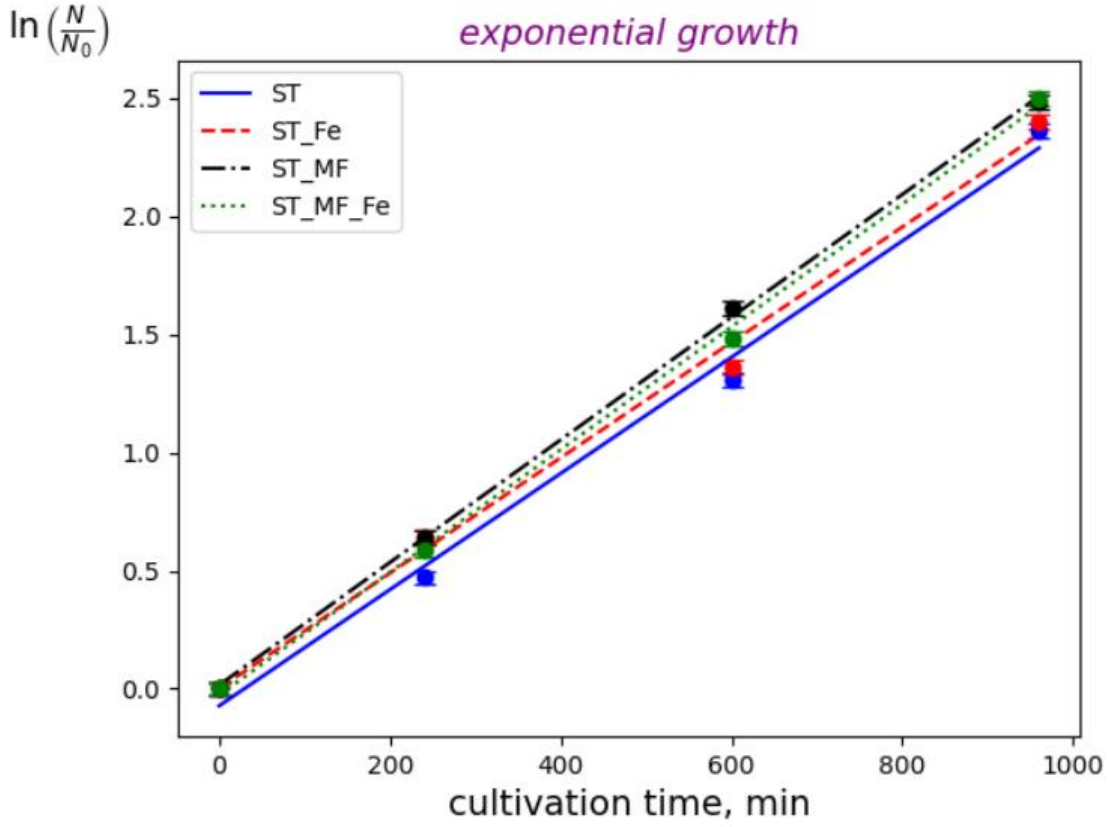


Рисунок 2.63 – Криві росту *E. coli* Nissle 1917, культивованих за різних умов: N – кількість клітинних кластерів на мл клітинної суспензії, $t \in [0; 16]$ годин – час культивування; N – кількість клітинних кластерів на мл клітинної суспензії, ST – стандартне середовище (контроль), ST+Fe - стандартне середовище з хелатами заліза без прикладення зовнішнього магнітного поля, ST+MF – стандартне середовище у зовнішньому 0,15 Тл, ST+MF+Fe - стандартне середовище з хелатами заліза в зовнішньому 0,15Тл. Лінії представляють апроксимацію експериментальних даних відповідно до моделі експоненціального зростання (16). Коефіцієнти моделі експоненціального зростання (16) є: τ_{ST}= 282 хв, τ_{ST_Fe}= 284 хв, τ_{ST_MF}= 267 хв, $\tau_{ST\_MF\_Fe}$= 267 хв, r_{ST}= 0,00246 хв$^{-1}$, r_{ST_Fe}= 0,00244 хв$^{-1}$, r_{ST_MF}= 0,0026 хв$^{-1}$, $r_{ST\_MF\_Fe}$= 0,0026 хв$^{-1}$.

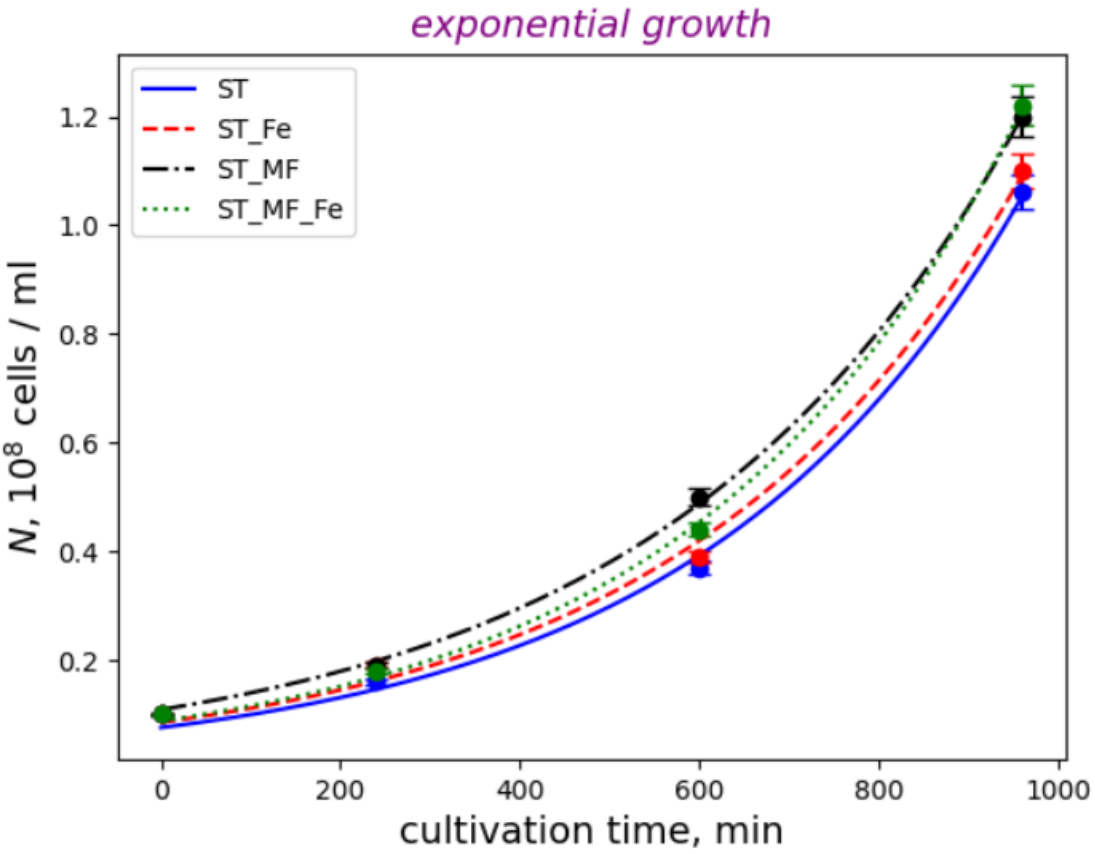


Рисунок 2.64 – Криві росту *E. coli* Nissle 1917, культивованих за різних умов: N – кількість клітинних кластерів на мл клітинної суспензії, $t \in [0; 16]$ годин – час культивування; N – кількість клітинних кластерів на мл клітинної суспензії, ST – стандартне середовище (контроль), ST+Fe - стандартне середовище з хелатами заліза без прикладення зовнішнього магнітного поля, ST+MF – стандартне середовище у зовнішньому 0,15 Тл, ST+MF+Fe - стандартне середовище з хелатами заліза в зовнішньому 0,15Тл. Лінії представляють апроксимацію експериментальних даних відповідно до моделі експоненціального зростання (16).

Далі розглянемо задачу про апроксимацію експериментальних даних кривих росту *E. coli* Nissle 1917, культивованих за різних умов, за функціями згідно кількох моделей [40] і обчислимо коефіцієнти, які входять у теоретичні формули кожної з моделей [40], в залежності від умов культивування.

В роботі [40] викладено огляд найбільш відомих моделей кривої росту бактерій. Прогнозне моделювання є перспективним напрямком мікробіології. Моделі використовуються для опису поведінки мікроорганізмів у різних фізичних або хімічних умовах як температура та pH води. Ці моделі дозволяють здійснювати прогнозування мікробної безпеки або терміну зберігання продуктів, виявлення критичних частин процесу виробництва та розподілу, а також оптимізувати виробництво. Щоб побудувати ці моделі, вимірювали часові залежності кількості клітин мікроорганізмів в процесі росту та здійснювали моделювання таких залежностей. Ріст бактерій часто показує фазу а, на якій питома швидкість росту починається зі значення нуль, а потім прискорюється до максимального значення в певний період часу, що призводить до затримки. Крім того, криві зростання містять заключну фазу, в якій темп зменшується і нарешті досягає нуля, так що досягається асимптотика. Коли криву росту визначають як логарифм кількості організмів, відображених у залежності від часу, ці зміни швидкості росту призводять до сигмоїдальної кривої, тобто спостерігається фаза затримки одразу після t = 0, за якою слідує експоненціальна фаза, а потім стаціонарна фаза. Щоб описати таку криву та звести виміряні

дані до обмеженої кількості важливих параметрів, дослідникам потрібні адекватні моделі. Існує низка моделей росту культури мікроорганізмі, такі як моделі Гомперца, Річардса, Стеннарда, Шнуте, логістична модель та інші [40]. Ці моделі описують лише кількість організмів і не включають споживання субстрату як це зроблено в моделі на основі рівняння Моно.

Спочатку здійснимо апроксимацію та обчислимо коефіцієнти b, c для двохпараметричної логістичної моделі, описаної в оглядовій роботі [40], вважаючи коефіцієнт n_0 визначеним з експерименту на основі даних про початкову кількість мікроорганізмів $N(0)$, Рисунок 2.65):

$$N(t) = \frac{n_0}{1+exp(b-ct)}.$$

Зауважимо, що коефіцієнт n_0 визначається через початкову кількість мікроорганізмів як $n_0 = N(0)(1 + exp(b))$.

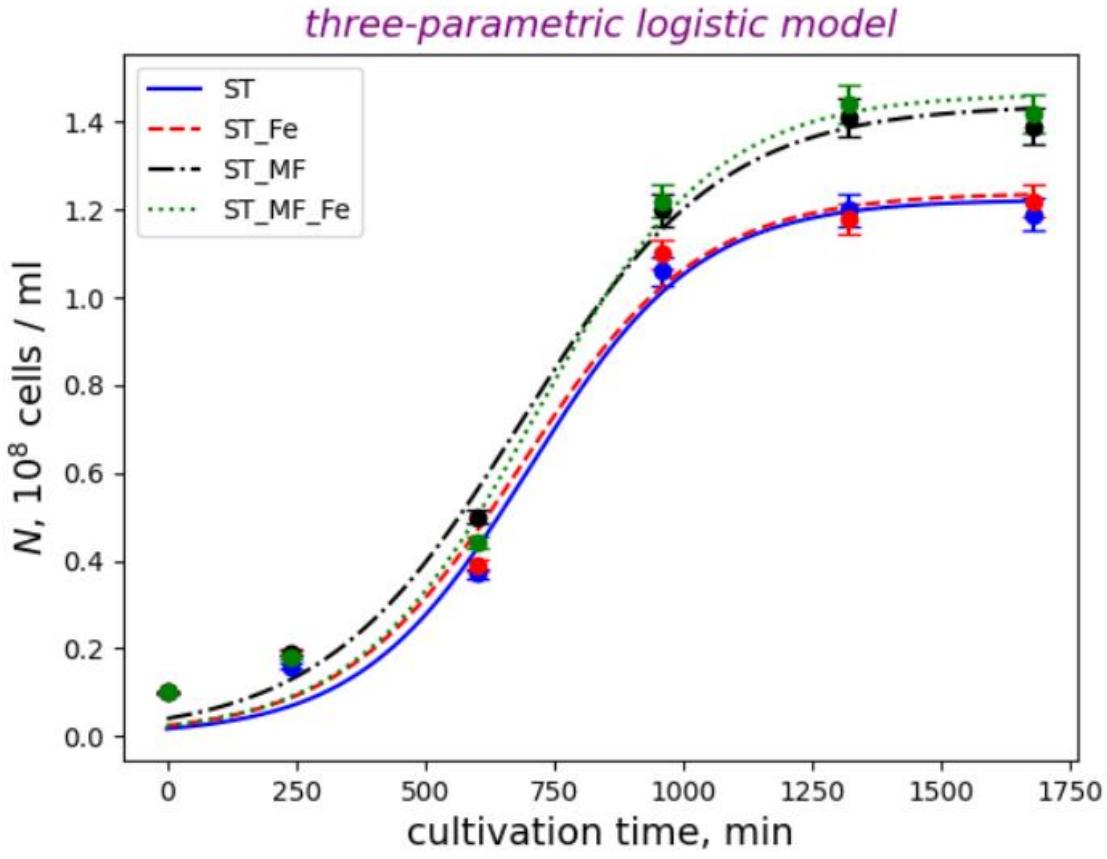


Рисунок 2.65 – Криві росту *E. coli* Nissle 1917, культивованих за різних умов: N – кількість клітинних кластерів на мл клітинної суспензії, t – час культивування; N – кількість клітинних кластерів на мл клітинної суспензії, ST – стандартне середовище (контроль), ST+Fe - стандартне середовище з хелатами заліза без прикладення зовнішнього магнітного поля, ST+MF – стандартне середовище у зовнішньому 0,15 Тл, ST+MF+Fe - стандартне середовище з хелатами заліза в зовнішньому 0,15Тл. Лінії представляють апроксимацію експериментальних даних відповідно до логістичної моделі [40]. Коефіцієнти двохпараметричної логістичної моделі ($N(t) = \frac{n_0}{1+exp(b-ct)}$ [40]) становлять: n_{0_ST}= 1,223 ·10 ^{8} клітин/мл, $n_{0\_ST\_Fe}$= 1,239 ·10 ^{8} клітин/мл, $n_{0\_ST\_MF}$= 1,44 ·10 ^{8} клітин/мл, $n_{0\_ST\_MF\_Fe}$= 1,464 ·10 ^{8} клітин/мл, b_{ST}= 4,29, b_{ST_Fe}= 3,9 6, b_{ST_MF}= 3,56, $b_{ST\_MF\_Fe}$= 4,12, c_{ST}= 0,0061 хв$^{-1}$, $n_{0\_ST\_Fe}$= 0,0058 хв$^{-1}$, c_{ST_MF}= 0,0052 хв$^{-1}$, $c_{ST\_MF\_Fe}$= 0,0058 хв$^{-1}$.

Також здійснимо апроксимацію та обчислимо коефіцієнти ν, k, τ, ν для чотирьохпараметричної моделі Річардса [40], вважаючи коефіцієнт a заданим (Рисунок 2.66):

$$N(t) = a\{1 + \nu \cdot exp[k(\tau - t)]\}^{(-1/\nu)} .$$

Коефіцієнт a визначається через початкову кількість мікроорганізмів та інші незалежні коефіцієнти моделі як $a = \frac{N(0)}{\{1+\nu \cdot exp[k\tau]\}^{(-1/\nu)}}$.

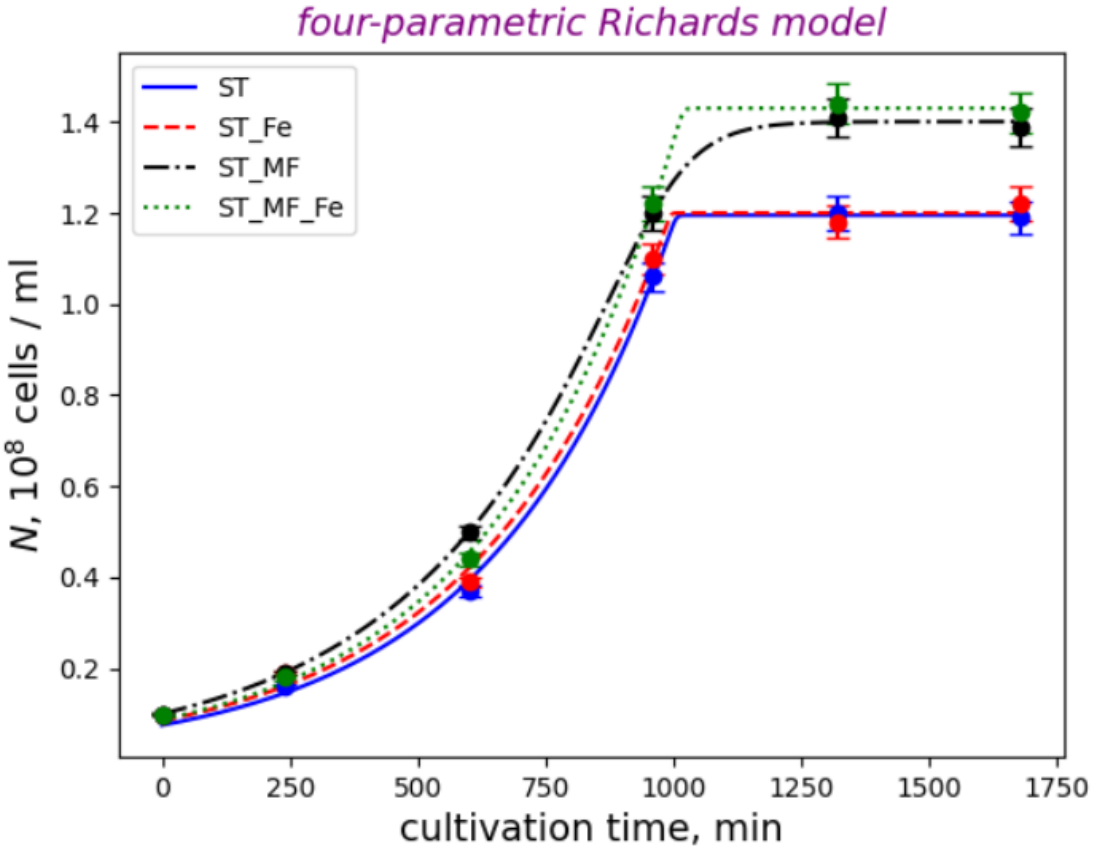


Рисунок 2.66 – Криві росту *E. coli* Nissle 1917, культивованих за різних умов: N – кількість клітинних кластерів на мл клітинної суспензії, t – час культивування; N – кількість клітинних кластерів на мл клітинної суспензії, ST – стандартне середовище (контроль), ST+Fe - стандартне середовище з хелатами заліза без прикладення зовнішнього магнітного поля, ST+MF – стандартне середовище у зовнішньому 0,15 Тл, ST+MF+Fe - стандартне середовище з хелатами заліза в зовнішньому 0,15Тл. Лінії представляють апроксимацію експериментальних даних за чотирьохпараметричною моделлю Річардса [40]. Коефіцієнти чотирьохпараметричної моделі Річардса ($N(t) = a\{1 + \nu \cdot exp[k(\tau - t)]\}^{(-1/\nu)}$ [40]) становлять a_{ST}= 1,195 ·10 ^{8} клітин/мл, a_{ST_Fe}= 1,2 ·10^8 клітин/мл, a_{ST_MF}= 1,4 ·10^8 клітин/мл, $a_{ST\_MF\_Fe}$= 1,43 ·10^8 клітин/мл, ν_{ST} = 107,17, ν_{ST_Fe} = 145,93, ν_{ST_MF}= 6,03, $\nu_{ST\_MF\_Fe}$ = 92,11, k_{ST}= 0,2942 хв$^{-1}$, k_{ST_Fe}= 0,388 хв$^{-1}$, k_{ST_MF}= 0,0161 хв$^{-1}$, $k_{ST\_MF\_Fe}$= 0,2522 хв$^{-1}$, τ_{ST}= 1059 хв, τ_{ST_Fe}= 1397 хв, τ_{ST_MF}= 58 хв, $\tau_{ST\_MF\_Fe}$= 908 хв.

Також здійснимо апроксимацію та обчислимо коефіцієнти a, l, k, p для чотирьохпараметричної моделлю Станнарда [40] (Рисунок 2.67):

$$N(t) = a\left\{1 + exp\left[-\frac{(l+kt)}{p}\right]\right\}^{(-p)} .$$

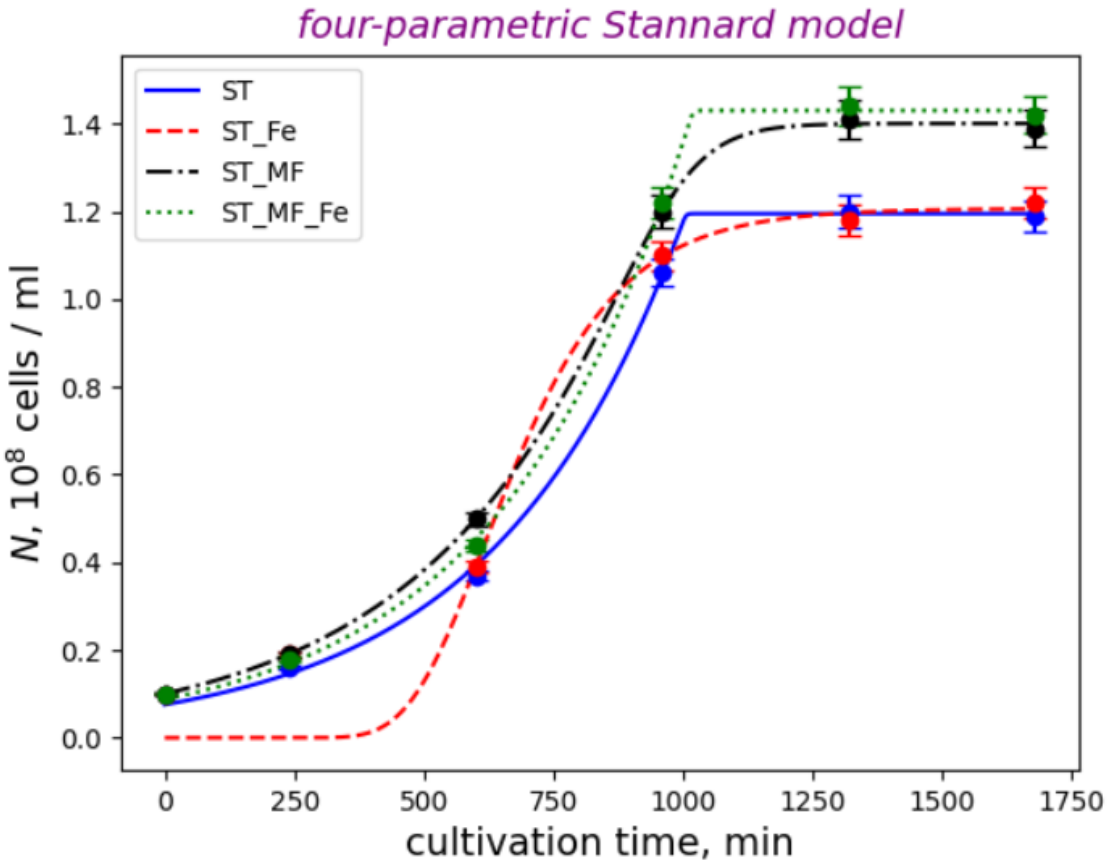


Рисунок 2.67 – Криві росту *E. coli* Nissle 1917, культивованих за різних умов: N – кількість клітинних кластерів на мл клітинної суспензії, t – час культивування; N – кількість клітинних кластерів на мл клітинної суспензії, ST – стандартне середовище (контроль), ST+Fe - стандартне середовище з хелатами заліза без прикладення зовнішнього магнітного поля, ST+MF – стандартне середовище у зовнішньому 0,15 Тл, ST+MF+Fe - стандартне середовище з хелатами заліза в зовнішньому 0,15Тл. Лінії представляють апроксимацію експериментальних даних за чотирьохпараметричною моделлю Станнарда [40]. Коефіцієнти чотирьохпараметричної моделі Стеннарда ($N(t) = a\left\{1 + exp\left[-\frac{(l+kt)}{p}\right]\right\}^{(-p)}$ [40]) є a_{ST}= 1,195 ·10^8 клітин/мл, a_{ST_Fe}= 1,206 ·10^8 клітин/мл, a_{ST_MF}= 1,4 ·10^8 клітин/мл, $a_{ST\_MF\_Fe}$= 1,43 ·10^8 клітин/мл, l_{ST}= -2,76, l_{ST_Fe}= 607,61, l_{ST_MF}= -2,63, $l_{ST\_MF\_Fe}$= -2,79, k_{ST}= 0,0027 хв$^{-1}$, k_{ST_Fe}= 2,5247 хв$^{-1}$, k_{ST_MF}= 0,0027 хв$^{-1}$, $k_{ST\_MF\_Fe}$= 0,0027 хв$^{-1}$, p_{ST}= 0,01, k_{ST_Fe}= 367,01, k_{ST_MF}= 0,17, $k_{ST\_MF\_Fe}$= 0,01.

Таким чином, на прикладі дослідження впливу умов культивування на динаміку росту культури мікроорганізмів показано, що засобами мови python можна здійснювати апроксимацію експериментальних даних різними функціями згідно різних відомих моделей. Оцінка якості кожної з моделей може бути здійснена візуально по графіках, методом найменших квадратів згідно виразу (15), або шляхом підрахунку коефіцієнта детермінації для апроксимації експериментальних даних згідно кожної моделі, які були описані вище.

Завдання та запитання до підрозділу «Пакети python для задач регресійного аналізу»

1. Для чого використовується регресійний аналіз?
2. Надайте визначення поняття медіана.
3. Що таке нахил лінії регресії?
4. Що означає intercept?
5. Який сенс має значення коефіцієнта кореляції?

6. Що означають stderr та intercept_stderr?
7. Дано списки даних для залежної (y) та незалежної (x) змінних:
x = [1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0]
y = [110, 107, 98, 77, 72, 67, 62, 59, 44, 31, 22, 19, 6]
Написати код, який будує графік залежної змінної від незалежної та розв'язує задачу лінійної регресії з використанням методу linregress(). Роздрукуйте атрибути, які повертає метод linregress(), округліть дані до трьох знаків після коми.

2.5. Пакет OpenCV та комп'ютерний зір

2.5.1. Поняття комп'ютерного зору та аналіз зображень

Комп'ютерний зір (Computer Vision) – це галузь штучного інтелекту, яка вивчає, як комп'ютери можуть інтерпретувати та розуміти візуальну інформацію із зображень або відео. Вона використовується для розв'язання різних завдань, пов'язаних з обробкою зображень та розпізнаванням об'єктів. Основні аспекти комп'ютерного зору включають:

1. Захоплення зображень – використання камер або інших сенсорів для отримання візуальних даних.
2. Попередня обробка – фільтрація, зменшення шуму та корекція кольору.
3. Витягнення ознак – використання алгоритмів для виділення особливих ознак із зображення, таких як краї, кутові точки або текстурні шаблони.
4. Аналіз зображень – використання різних методів та моделей машинного навчання для розпізнавання об'єктів, класифікації зображень, виявлення об'єктів на зображенні, вимірювання відстаней і багато іншого.

Аналіз зображень – це процес обробки та інтерпретації візуальних даних, включаючи виявлення об'єктів, вимірювання їх параметрів та винесення рішень на основі цієї інформації. Цей процес може бути застосований в багатьох галузях, включаючи науку, промисловість, медицину тощо. Для аналізу зображень використовуються різні методи, включаючи фільтрацію, виокремлення об'єктів, обчислення статистичних параметрів та використання алгоритмів машинного навчання для вирішення завдань, пов'язаних з аналізом зображень.

Ці дві області, комп'ютерний зір і аналіз зображень, є важливими для розвитку багатьох сучасних технологій і застосовуються в різних наукових та практичних задачах.

2.5.2. Загальні принципи роботи пакету OpenCV

OpenCV (Open Source Computer Vision) – це потужна бібліотека з відкритим вихідним кодом, яка спеціалізується на обробці та аналізі зображень і відео. Вона містить сотні алгоритмів комп'ютерного бачення і є надзвичайно популярною серед дослідників, розробників та інженерів у всьому світі [41, 42].

Першу версію OpenCV було розроблено компанією Intel у 1998 році, і вона була написана мовами програмування C та C++ [42]. З тих пір бібліотека розвивалася та вдосконалювалася завдяки активній участі розробників у спільноті та підтримці великої кількості користувачів.

OpenCV підтримує багато мов програмування, що робить її доступною для широкого кола розробників. Основними мовами для роботи з пакетом є C/C++, python, Java, а також інші мови, такі як Ruby та Matlab.

Бібліотека OpenCV пропонує широкий спектр функцій і алгоритмів для обробки зображень та відео. Вона дозволяє виконувати операції, такі як зчитування та запис зображень, фільтрація, знаходження країв, виявлення об'єктів, вимірювання розмірів, розпізнавання обличчя та багато інших.

OpenCV також має підтримку для інтеграції з іншими бібліотеками машинного навчання, що дозволяє розробникам використовувати сучасні алгоритми для вирішення завдань комп'ютерного зору.

Ця бібліотека знаходить застосування в різних галузях, включаючи комп'ютерний зір для автономних автомобілів, розпізнавання обличчя у фотографіях та відео, відслідковування руху об'єктів, віртуальну реальність, медичну діагностику, робототехніку і багато інших.

Пакет OpenCV підтримується великою активною спільнотою розробників і дослідників, яка постійно вносить внески у розвиток бібліотеки, створює нові модулі та алгоритми і надає підтримку користувачам через форуми та різні онлайн-ресурси.

Дана бібліотека поширюється за ліцензією BSD (Berkeley Software Distribution license), що означає, що її можна вільно і безкоштовно використовувати як в відкритих проєктах з відкритим кодом, так і в закритих, комерційних проєктах. Єдина вимога ліцензії – наявність в супроводжуючих матеріалах копії ліцензії OpenCV.

Узагальнюючи, OpenCV є потужним інструментом для роботи з комп'ютерним баченням і аналізом зображень, і вона продовжує залишатися основним ресурсом для великої кількості проєктів і досліджень у сфері обробки візуальної інформації. Більш детально про OpenCV можна дізнатись на їх офіційному сайті http://opencv.org [43], а документацію можна знайти за посиланням https://docs.opencv.org/ [41].

2.5.3. Структура пакету OpenCV

OpenCV має модульну структуру, що означає, що пакет включає кілька спільних або статичних бібліотек. Доступні такі основні модулі:

Основна функціональність (**core**) – компактний модуль, що визначає базові структури даних, включаючи багатовимірний масив Mat і базові функції, які використовуються всіма іншими модулями.

Обробка зображень (**imgproc**) – модуль обробки зображень, який включає лінійну та нелінійну фільтрацію зображень, геометричні перетворення зображення (зміна розміру, афінне та перспективне викривлення, загальне перевідображення на основі таблиці), перетворення простору кольорів, гістограми тощо.

Аналіз відео (**video**) – модуль аналізу відео, який включає алгоритми оцінки руху, віднімання фону та відстеження об’єктів.

Калібрування камери та 3D-реконструкція (**calib3d**) – базові алгоритми геометрії кількох ракурсів, калібрування одиночної та стереокамери, оцінка позиції об’єкта, алгоритми стереовідповідності та елементи 3D-реконструкції.

Функції для 2D обробки (**features2d**) – детектори, дескриптори та відповідники дескрипторів збігів.

Виявлення об'єктів (**objdetect**) – виявлення об'єктів і екземплярів попередньо визначених класів (наприклад, обличчя, бактерії, люди, машини і так далі).

Графічний інтерфейс високого рівня (**highgui**) – простий у використанні інтерфейс для простих можливостей інтерфейсу користувача.

Захоплення та виведення відео (**videoio**) – простий у використанні інтерфейс для захоплення відео та відеокодеків.

2.5.4. Поєднання python та OpenCV

Python є однією з найпопулярніших мов програмування для роботи з бібліотекою OpenCV. Використання python в поєднанні з OpenCV робить комп'ютерне бачення і аналіз зображень більш доступними та зручними для розробників та дослідників. Основні переваги використання python при роботі з OpenCV включають:

- легку установку, оскільки python має просту систему установки та оновлення пакетів, а OpenCV можна встановити через пакетний менеджер pip чи за допомогою розділу python packages у PyCharm.

- простий та зрозумілий синтаксис, що сприяє швидкому розвитку програм для обробки зображень.
- зручність роботи з багатовимірними масивами даних завдяки бібліотекам, таким як NumPy.
- можливість візуалізації результатів за допомогою бібліотеки Matplotlib.
- інтеграція з іншими бібліотеками машинного навчання, такими як TensorFlow та PyTorch.
- розширені можливості машинного навчання для розпізнавання об'єктів та класифікації зображень.
- крос-платформеність python і OpenCV, яка дозволяє працювати на різних операційних системах.

Використання python разом з OpenCV робить розробку рішень комп'ютерного зору більш приємною та ефективною завданням і відкриває широкий спектр можливостей для аналізу та обробки візуальної інформації.

Важливим аспектом використання python при роботі з пакетом OpenCV є те, що вона може виявити обмеження щодо продуктивності, особливо при спробі використати цикли для обробки матриць зображень. Тому рекомендується уникати самостійного написання python-коду для обробки растрових зображень і, замість цього, використовувати вбудовані функції, які надає пакет OpenCV.

Пакет OpenCV містить в собі широкий функціонал, якого зазвичай вистачає для вирішення різних завдань обробки зображень. При цьому, використовуючи функціонал бібліотеки, Ви працюєте з обгорткою над оригінальним кодом, написаним на мовах програмування C/C++, за допомогою python API (Application Programming Interface). Це означає, що при правильній архітектурі проєкту комп'ютерного зору, Ваш python-код може бути досить продуктивним і надійним, майже на рівні з кодом, написаним на C/C++.

2.5.5. Початок роботи із OpenCV

Оскільки OpenCV не є стандартною бібліотекою python, то її слід завантажити:

- через розділ python packages у PyCharm (Рисунок 2.68):

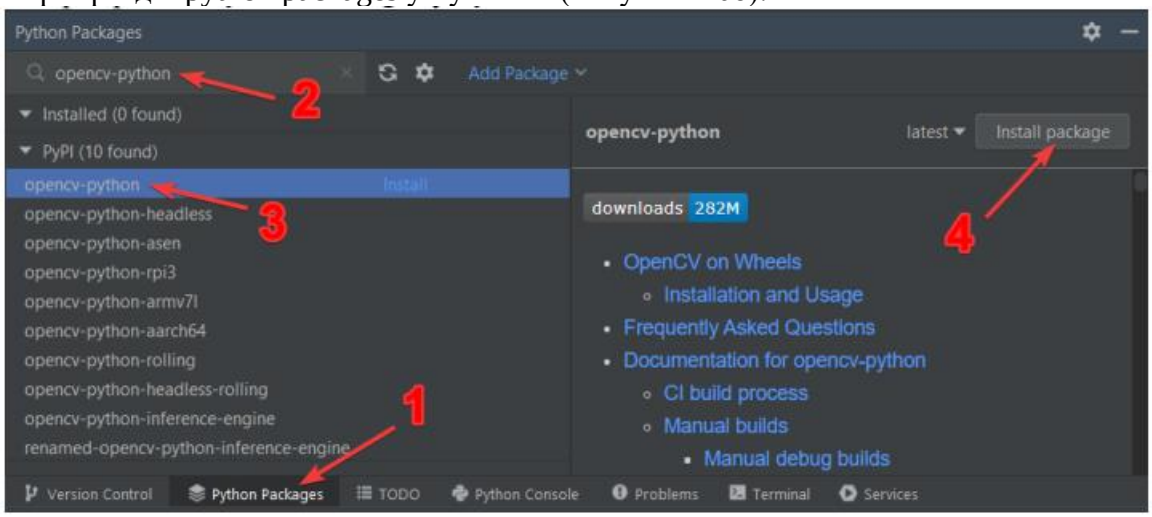


Рисунок 2.68 – Схема завантаження пакету opencv-python через python packages у PyCharm

- через пакетний менеджер pip:

Відкрийте консоль (термінал) та напишіть команду «pip install opencv-python».

Для імпорту до файлу функціоналу цієї бібліотеки слід вказати назву cv2, яку за потреби можна замінити на псевдонім (наприклад, cv). Найпростіший код для відкриття зображення через OpenCV матиме вигляд:

```
import cv2
# Читання зображення у матрицю
img = cv2.imread('bacterium.png')
# Відображення картинки у новому вікні
cv2.imshow('Bacterium', img)
cv2.waitKey(0)
```

Результат роботи коду (Рисунок 2.69):

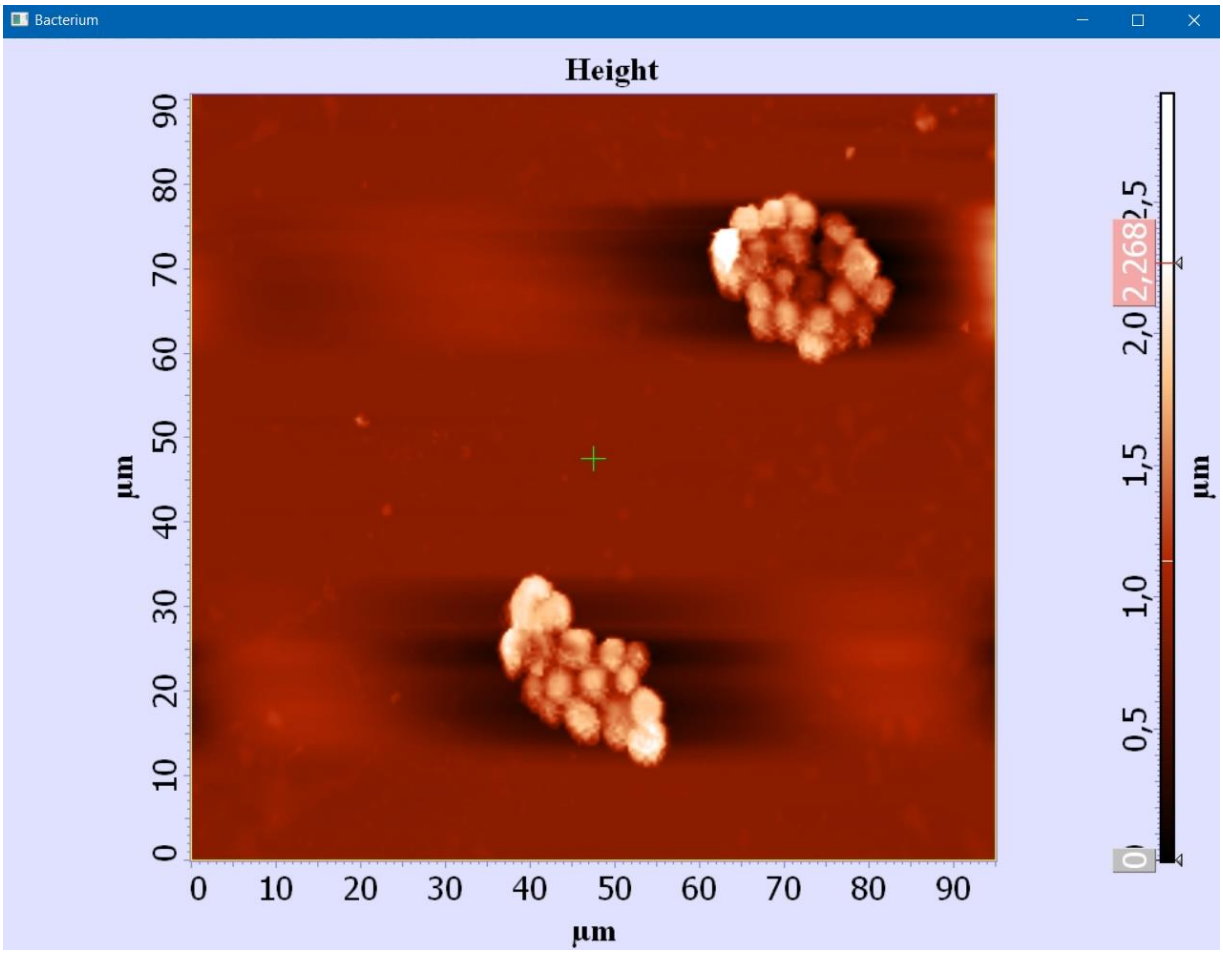


Рисунок 2.69 – Результат роботи коду із попереднього прикладу

2.5.6. Робота із зображеннями у OpenCV

Зображення з точки зору OpenCV – це матриця чисел, де кожне число представляє інтенсивність світла (яскравість) пікселя на певній позиції. Зображення складається з пікселів, і кожен піксель має свої координати (рядок та стовпець) та значення інтенсивності світла.

У більш загальному контексті, зображення може бути кольоровим (тривимірна матриця) або в чорно-білому варіанті (двовимірна матриця). Кольорові зображення зазвичай кодуються за допомогою трьох каналів: червоного (R), зеленого (G) і синього (B), де кожен канал відображає інтенсивність світла для відповідного кольору. Чорно-білі зображення мають тільки один канал, який відображає яскравість пікселів.

Колір кожного пікселя задається за допомогою трьох значень. Ці компоненти зазвичай можуть мати значення від 0 до 255. Наприклад, піксель з значенням RGB(255, 0, 0) буде червоним, піксель з значенням RGB(0, 255, 0) буде зеленим, а піксель з значенням RGB(0, 0, 255) буде синім. Проте з історичних причин OpenCV при читанні використовує інший

подібний кольоровий простір – BGR (Blue, Green, Red), тобто зміни лише за порядком запису інформації у пам’яті.

Також зображення може мати додатковий канал, який відповідає за прозорість пікселя, – альфа-канал. Його значення також змінюються в діапазоні від 0 до 255. Значення 0 відповідає повній прозорості, а при значенні 255 даний піксель буде повністю непрозорим.

Пікселі в зображенні розташовані в рядках і стовпцях. Рядки називаються віссю X, а стовпці – віссю Y. Початковий піксель в зображенні знаходиться в лівому верхньому куті, а останній піксель - в правому нижньому куті.

Приклад коду, що описує розмір та кількість каналів, а також обрізає зображення подібно до зрізу матриць:

```
import cv2

img = cv2.imread('bacterium.png')
# Опис зображення
print(f'Розмір картинки: {img.shape[0:2]}, кількість каналів у вихідного зображення: {img.shape[2]}')
# Вихідне зображення
cv2.imshow('Img', img)
# Кадрування зображення подібно до матриць
cv2.imshow('New', img[50:700, 200:712])
cv2.waitKey(0)
```

Результат роботи коду (Рисунок 2.70):

```
Розмір картинки: (471, 607), кількість каналів у вихідного зображення: 3.
```

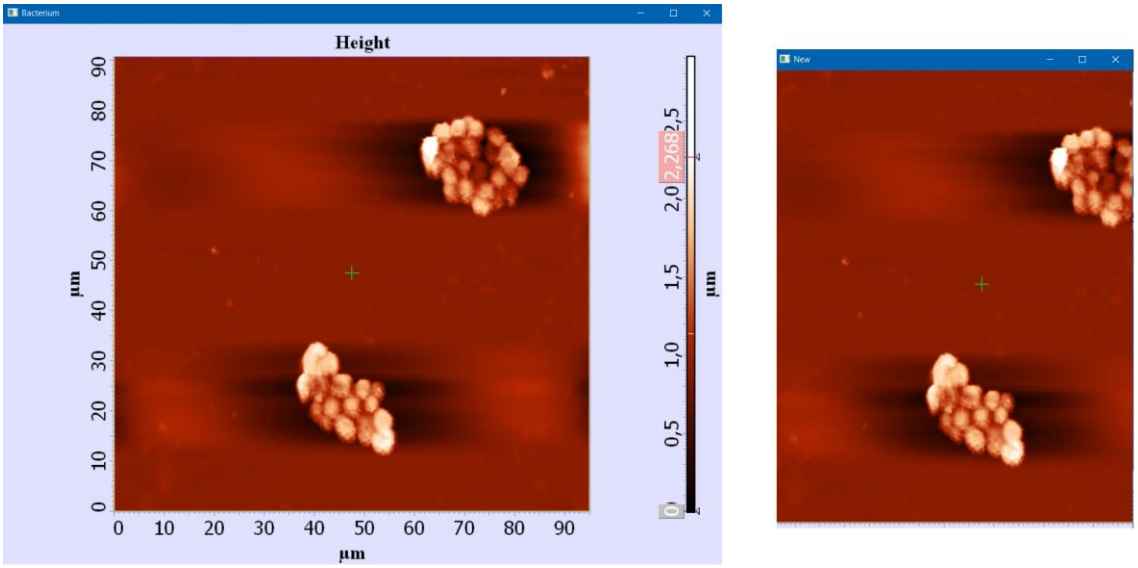


Рисунок 2.70 – Результат роботи коду із попереднього прикладу

2.5.6.1. Базова обробка зображень

OpenCV надає широкий спектр функцій для роботи з зображеннями. Ці функції можна використовувати для обробки зображень, таких як зміна розміру, поворот, обрізка, згладжування, різкість, виявлення країв, розпізнавання об'єктів і багато іншого.

Ось деякі з найпоширеніших функцій OpenCV для роботи з зображеннями:

- cv2.imread(filename, flag) – завантажує зображення з файлу filename; параметр flag відповідає за режим читання: cv2.IMREAD_COLOR (читання кольорового зображення),

cv2.IMREAD_UNCHANGED (читання файлу без змін, включаючи альфа-канал), cv2.IMREAD_GRAYSCALE (читання у форматі чорно-білого зображення).

- cv2.imshow(winname, img) – відображає зображення img на екрані у вікні з назвою winname.
- cv2.waitkey(n) – функція, що використовується для вказання часу затримки, наприклад, щоб показати зображення певний проміжок часу. Параметр n відповідає кількості мілісекунд затримки. Якщо n = 0, то кадр буде відображатись поки не закрити вікно. Також функція повертає числове значення кнопок, які натискає користувач, що використовується разом з умовними конструкціями.
- cv2.destroyAllWindows() – закриває всі відкриті програмою вікна.
- cv2.imwrite(filename, img) – зберігає зображення img в файлі з назвою filename.
- cv2.resize(img, dsize, fx, fy, interpolation) – змінює розмір зображення img до розмірів dsize (масив із двох цілих чисел) або за допомогою коефіцієнтів fx та fy (0.5 зменшить зображення вдвічі, а 2 – збільшить). Переважними методами інтерполяції (interpolation) є cv2.INTER_AREA для стискання та cv2.INTER_CUBIC (повільний) та cv2.INTER_LINEAR (за замовчуванням) для масштабування.
- cv2.rotate(img, rotateCode) – повертає зображення img на відповідний кут, який задається через константу із пакету cv2 (наприклад, cv2.ROTATE_180).

Приклад повноцінного коду, який трансформує та записує змінене зображення:

```
import cv2
# Читання зображення у матрицю
img = cv2.imread('bacterium.png')
# Відображення картинки у новому вікні
cv2.imshow('bacterium', img)
# Зменшення зображення у 2 рази
img_resized = cv2.resize(img, None, fx=0.5, fy=0.5,
interpolation=cv2.INTER_CUBIC)
# Відображення зменшеної картинки у новому вікні із заголовком
Resized
cv2.imshow('Resized', img_resized)
# Поворот зменшеного зображення на 90 градусів проти годинникової
стрілки
img_rotated = cv2.rotate(img_resized, cv2.ROTATE_180)
# Відображення зменшеної та повернутої на 90 градусів картинки
cv2.imshow('Resized and rotated', img_rotated)
if cv2.waitKey(0) == ord('q'):  # Для закриття треба натиснути q
    cv2.destroyAllWindows()
# Запис зміненого зображення у файл
cv2.imwrite('bacterium_changed.png', img_rotated)
```

Результат роботи коду, окрім збереження файлу bacterium_changed.png у поточну директорію (Рисунок 2.71):

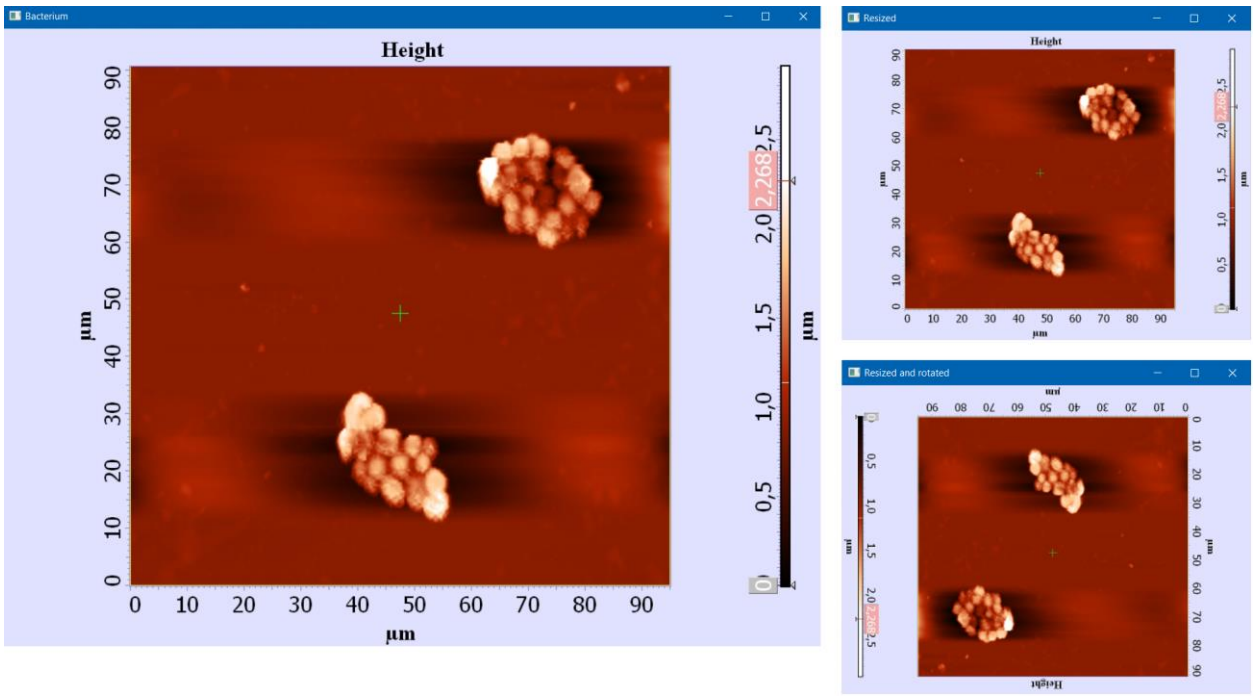


Рисунок 2.71 – Результат роботи коду із попереднього прикладу

2.5.6.2. Кольоровий простір зображення та їх види

При роботі із зображеннями часто може виникати потреба виділити певні кольори. Якщо цільовий діапазон кольорів досить широкий (наприклад, всі блакитні відтінки), то використання BGR/RGB простору може бути не зручним. Натомість у таких випадках використовують інші – HSV, HLS, Lab тощо. Для перетворення зображення потрібно використати функцію cv2.cvtColor(img, code). Код перетворення є константою, яку можна обрати із доступних у пакеті OpenCV, наприклад, cv2.COLOR_BGR2HSV, cv2.COLOR_BGR2HLS, cv2.COLOR_BGR2Lab, cv2.COLOR_BGR2GRAY тощо.

Для виділення діапазонів кольорів використовується функція cv2.inRange(img, lowerb, upperb), де lowerb – кортеж із значеннями нижньої межі, а upperb – кортеж із значеннями верхньої межі. Тобто lowerb=(0, 0, 0) і upperb=(255, 255, 255) не буде накладати жодних обмежень на зображення.

Накладання отриманої маски від функції inRange на матрицю-зображення відбувається за допомогою логічної операції "і": cv2.bitwise_and(src1, src2, mask) – повертає матрицю побітової кон'юнкції (AND) між двома зображеннями src1 та src2, помножену на mask, яка містить значення 0 чи 1.

Приклад роботи коду з використанням перетворень зображення у інші кольорові простори:

```
import cv2
# Читання зображення у матрицю
img = cv2.imread('bacterium.png', cv2.IMREAD_COLOR)
# Відображення сірої картинки
cv2.imshow('Unchanged', img)
# Зміна кольорового простору на сірий
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Відображення сірої картинки
cv2.imshow('Gray', img_gray)
# Перетворення зображення у HSV (Hue Saturation Value)
img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
# Створення маски, де значення лежать в межах нижньої та верхньої границь
```

```
mask = cv2.inRange(img_hsv, (0, 221, 0), (10, 255, 255))
# Накладання маски шляхом логічної операції "і"
img_red = cv2.bitwise_and(img, img, mask=mask)
# Відображення червоних ділянок картинки
cv2.imshow('Red', img_red)
if cv2.waitKey(0) == ord('q'):  # Для закриття треба натиснути q
    cv2.destroyAllWindows()
```

Результат роботи коду (Рисунок 2.72):

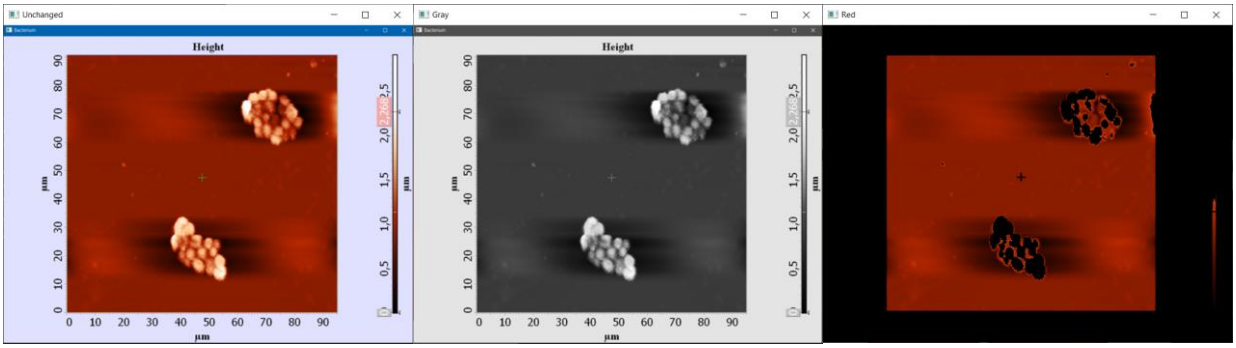


Рисунок 2. 72 – Результат роботи коду із попереднього прикладу

2.5.6.3. Застосування порогового розподілу

Функція cv2.threshold() є однією з функцій бібліотеки OpenCV і використовується для порогової бінаризації зображень. Порогова бінаризація – це процес перетворення зображення в чорно-біле (бінарне) зображення, де кожен піксель може бути або чорним (значення 0) або білим (максимальне значення, зазвичай 255), в залежності від того, чи задовольняє піксель певному пороговому значенню.

Після виклику цієї функції, вона повертає два значення: порогове значення, що було використано, та бінарне зображення.

1. src – вхідне зображення – вхідний масив src (багатоканальний, 8-бітний або 32-розрядний з плаваючою точкою)
2. thresh – порогове значення кольору.
3. maxval – максимальне значення (зазвичай 255).
4. type – тип бінаризації, який визначає, як обробляти пікселі, які задовольняють порогу. Ви можете використовувати різні типи бінаризації, такі як cv2.THRESH_BINARY (стандартна бінаризація), cv2.THRESH_BINARY_INV (інвертована бінаризація), cv2.THRESH_TRUNC (всі пікселі, які перевищують поріг, залишаються без змін), і багато інших.

Приклад використання порогових значень:

```
import cv2
# Зчитування зображення
img = cv2.imread('gray.jpg')
# Відображення вихідного зображення
cv2.imshow('Img', img)
# Застосування порогу: світліші пікселі стають білими, а темніші – чорними
ret, img = cv2.threshold(img, 71, 255, cv2.THRESH_BINARY)
# Друк ret
print('Використане порогове значення:', ret)
# Відображення бінарної картинки
```

```
cv2.imshow('Threshold', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Результат роботи коду (Рисунок 2.73):
Використане порогове значення: 71.0

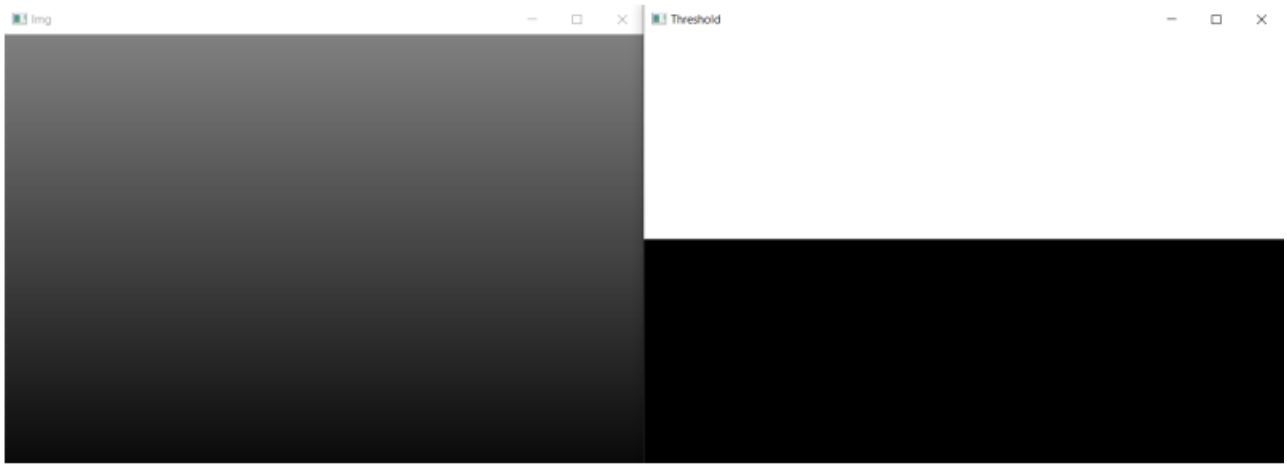


Рисунок 2.73 – Результат роботи коду із попереднього прикладу

2.5.6.4. Детекція країв на зображенні

Функція cv2.GaussianBlur() в бібліотеці OpenCV використовується для застосування фільтра Гауса до зображення. Це важливо для зменшення шуму та згладжування зображення перед подальшими обробками. Параметри цієї функції:

1. src – вхідне зображення.
2. ksize – розмір ядра фільтра Гауса. Це кортеж і визначає розмір фільтра. Наприклад, (5, 5) вказує на фільтр розміром 5x5.
3. sigmaX – стандартне відхилення по осі X. Воно визначає, як сильно буде згладжуване зображення по горизонталі. Зазвичай встановлюється на ненульове значення.
4. sigmaY (необов'язковий) – стандартне відхилення по осі Y. Воно визначає, як сильно буде згладжуване зображення по вертикалі. Якщо він не вказаний, використовується таке ж значення, як sigmaX.

Функція cv2.Canny() використовується для створення границь на зображенні за допомогою алгоритму Кенні. Параметри цієї функції:

1. image – вхідне зображення, на якому потрібно виявити границі.
2. threshold1 і threshold2 – дві порогові величини для визначення меж границь. Границі знаходяться шляхом відсіювання пікселів зі значеннями градієнту між threshold1 та threshold2.
3. apertureSize – розмір ядра для обчислення градієнту (зазвичай використовується 3).

Ці функції допомагають в обробці та аналізі зображень, зокрема в зменшенні шуму та виявленні границь, і є важливими для багатьох завдань у комп'ютерному зорі та обробці зображень. Варто також до детектування скористатися розмиттям, щоб зменшити кількість країв. Приклад коду із використанням даних функцій:

```
import cv2
# Читання зображення у матрицю
img = cv2.imread('bacterium.png', cv2.IMREAD_COLOR)
# Зменшення розмірів зображення вдвічі
img = cv2.resize(img, None, fx=0.5, fy=0.5)
```

```
# Відображення незміненого зображення
cv2.imshow('Unchanged', img)
# Застосування розмиття по Гаусу
img_blurred = cv2.GaussianBlur(img, (5, 5), 2.5)
# Відображення розмитого зображення
cv2.imshow('Blurred', img_blurred)
# Застосування детектора країв до вихідного зображення
img_canny = cv2.Canny(img, 0, 128)
# Відображення картинки із детектованими краями
cv2.imshow('Canny', img_canny)
# Застосування детектора країв до розмитого зображення
img_blurred_canny = cv2.Canny(img_blurred, 0, 128)
# Відображення картинки із детектованими краями
cv2.imshow('Blurred+Canny', img_blurred_canny)
# Відображення до закриття вікна
cv2.waitKey(0)
```

Результат роботи коду (Рисунок 2.74):

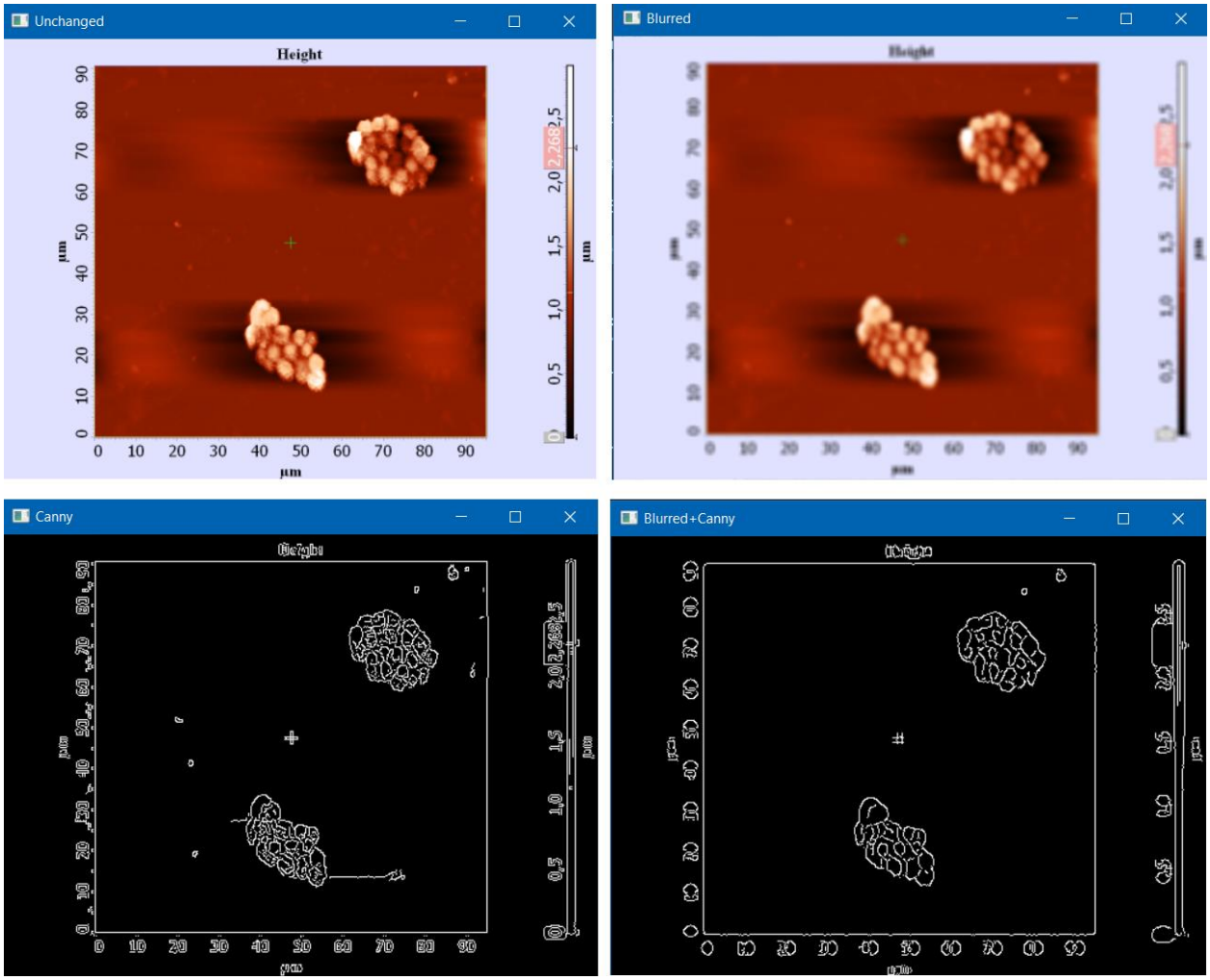


Рисунок 2.74 – Результат роботи коду із попереднього прикладу

2.5.6.5. Складна робота з контурами

Створення контурів зазвичай не має практичного застосування без подальшої обробки. Отримане від функції cv2.Canny() бінарного зображення (лише значення 0 або 255) використовується для подальшого дослідження за допомогою функції cv2.findContours().

Функція cv2.findContours() в бібліотеці OpenCV використовується для знаходження контурів на бінарних зображеннях. Контур – це замкнений шлях, який обводить об'єкт на зображенні. Ця функція корисна для виявлення об'єктів або обрисів на зображеннях та подальшої обробки їхніх характеристик, таких як площа, довжина контуру і інші. Основні параметри функції cv2.findContours():

1. image – вхідне бінарне зображення, на якому шукатимуться контури.
2. mode – режим пошуку контурів. Це один з режимів пошуку, таких як cv2.RETR_EXTERNAL (знаходить тільки зовнішні контури), cv2.RETR_LIST (знаходить всі контури і зберігає їх у списку) та інші. Вибір режиму залежить від Вашого завдання.
3. method – метод апроксимації контурів. Зазвичай використовується cv2.CHAIN_APPROX_SIMPLE, який видаляє всі зайві точки та зберігає тільки вершини контурів. Інший варіант - cv2.CHAIN_APPROX_NONE, який зберігає всі точки контурів.

Після виклику функції cv2.findContours() повертається список контурів, знайдених на зображенні, а також інформацію про її ієрархію, якщо це вказано.

Після створення списку контурів їх можна відобразити. Функція cv2.drawContours() використовується для малювання контурів на зображенні. Вона дозволяє відображати контури, знайдені за допомогою функції cv2.findContours(), на вихідному зображенні або на порожньому зображенні за допомогою заданих параметрів. Основні параметри функції cv2.drawContours():

1. image – зображення, на якому будуть малюватися контури. Це може бути оригінальне зображення або порожнє.
2. contours – список контурів, які Ви хочете намалювати. Цей список зазвичай отримується в результаті виклику функції cv2.findContours().
3. contourIdx – індекс контура в списку contours, який потрібно намалювати. Якщо значення цього параметра встановлено на -1, то всі контури з списку будуть намальовані.
4. color – колір, яким будуть намальовані контури. Це може бути тривимірний кортеж, де кожен елемент відповідає значенням кольору BGR.
5. thickness – товщина лінії, якою будуть намальовані контури. Зазвичай це додатне ціле число. Якщо Ви встановите його на -1, то контур буде залитий кольором, а не намальований лінією.

Після виклику функції cv2.drawContours(), контури будуть намальовані на вказаному зображенні з вказаними параметрами.

Крім зображення контура, можна використати дані від cv2.findContours() для різних розрахунків. Наприклад, функція cv2.moments() використовується для обчислення моментів зображення, які є статистичними характеристиками зображення або контура. Моменти дозволяють отримувати різні важливі параметри об'єктів на зображенні, такі як центр мас, площа, орієнтація, інерційні моменти тощо. Вони широко використовуються в аналізі зображень та обробці зображень. Основним параметром функції є вхідне зображення або контур, для якого потрібно обчислити моменти. Це може бути бінарне зображення або контур, представлений як список точок (кожна точка – це координати (x, y)).

Функція cv2.moments() повертає словник, який містить різні моменти, такі як:

- 'm00' – загальна площа об'єкта або контура.
- 'm10' – сума x-координат пікселів усіх точок об'єкта або контура.
- 'm01' – сума y-координат пікселів усіх точок об'єкта або контура.
- 'm20' – інерційний момент другого порядку за віссю x.
- 'm02' – інерційний момент другого порядку за віссю y.
- 'm11' – інерційний момент першого порядку, зв'язаний з осями x та y.
- 'mu20' – центральний інерційний момент другого порядку за віссю x.

- 'mu02' – центральний інерційний момент другого порядку за віссю y.

Ці моменти можна використовувати для розрахунків і аналізу об'єктів на зображенні, таких як визначення центру мас об'єкта, розрахунок орієнтації об'єкта, визначення площі об'єкта і багатьох інших параметрів.

Простішими діями із даними про контури є визначення їх довжини та площі. Функції cv2.arcLength() і cv2.contourArea() в бібліотеці OpenCV використовуються для обчислення характеристик контурів, знайдених на зображенні.

Функція cv2.arcLength() призначена для обчислення довжини контура або довжини його замкненого об'єкта. Основний параметр contour – це контур, для якого потрібно обчислити довжину. Параметр closed – це логічне значення (True або False), яке вказує, чи є контур замкненим. Якщо closed встановлено в True, то функція обчислить довжину замкненого контура, який утворює замкнену фігуру, наприклад, коло або прямокутник. Якщо closed встановлено в False, то функція обчислить довжину відкритого контура, наприклад, ламаної лінії.

Повертається числове значення – довжина контура або лінії відповідно до вказаного параметра closed.

Функція cv2.contourArea() призначена для обчислення площі контура. Параметр contour – це контур, для якого потрібно обчислити площу. Параметр oriented – це логічне значення (True або False), яке вказує, чи потрібно враховувати орієнтацію площі контура. Якщо воно True, функція повертає значення площі зі знаком залежно від орієнтації контуру (за або проти годинникової стрілки). За допомогою цієї функції можна визначити орієнтацію контуру, взявши знак області. За замовчуванням параметр має значення False, що означає, що повертається абсолютне значення.

Повертається числове значення - площа контура.

Приклад складної, комплексної роботи із контурами:

```
import cv2

img = cv2.imread('bacterium.png')
# Перетворення у відтінки сірого
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Перетворення у бінарне зображення
edged = cv2.Canny(gray, 240, 255)
# Визначення контурів
contours, hierarchy = cv2.findContours(edged, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
print("Кількість знайдених контурів =", len(contours))
# Створення зелених контурів на зображенні (-1 означає
відображення всіх контурів)
cv2.drawContours(img, contours, -1, (0, 255, 0), 2)
# Створення червоного контуру навколо контуру під номером 89
cv2.drawContours(img, contours, 89, (0, 0, 255), 3)
# Розрахунок моментів для контуру під номером 89
moments = cv2.moments(contours[89])
# Визначення центру мас
cx = int(moments['m10']/moments['m00'])
cy = int(moments['m01']/moments['m00'])
# Зображення центру мас як синьої крапки
cv2.circle(img, (cx, cy), radius=0, color=(255, 0, 0),
thickness=5)
# Відображення картинки контурами та центром мас
```

```
cv2.imshow('Contours', img)
print('Довжина контура під номером 89:',
round(cv2.arcLength(contours[89], closed=True), 0), 'px')
print('Площа контура під номером 89:',
round(cv2.contourArea(contours[89]), 0), 'px^2')
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Результат роботи коду (Рисунок 2.75):

```
Кількість знайдених контурів = 217
Довжина контура під номером 89: 283.0 px
Площа контура під номером 89: 28.0 px^2
```

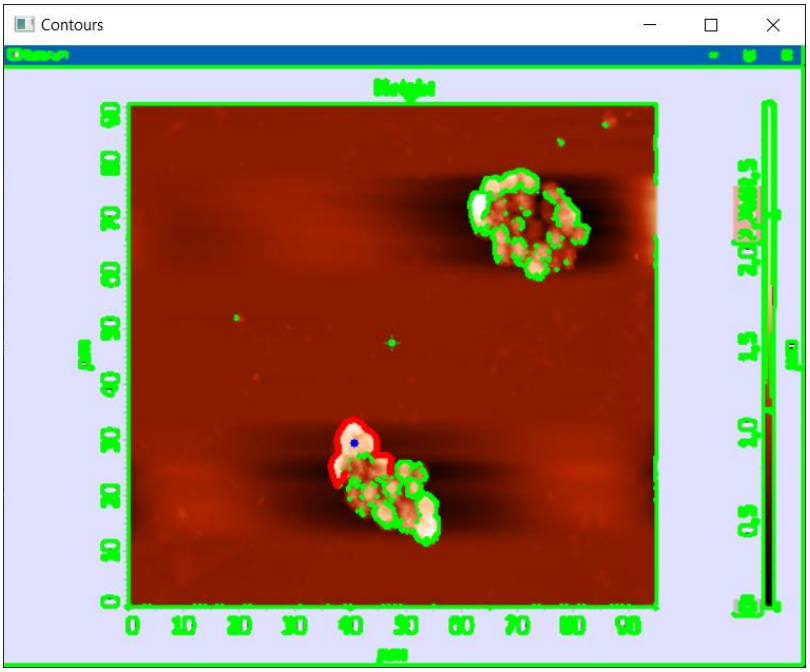


Рисунок 2.75 – Результат роботи коду із попереднього прикладу

2.5.6.6. Детектування об’єктів на зображенні

Однією з найвідоміших задач комп’ютерного бачення є розпізнавання обличчя або будь-яких інших об’єктів. Клас для завантаження та використання класифікаторів каскадів для виявлення об'єктів на зображеннях має назву cv2.CascadeClassifier. Єдиним параметром, який потрібно передати в конструктор класу є filename – шлях до файлу, в якому збережено навчений класифікатор каскаду. Цей файл містить інформацію про структуру та параметри каскаду, який використовується для виявлення об'єктів.

Проте безпосередньо для розпізнання використовується інша функція, що повертає набір цілих чисел (детальніше далі у поясненні функції rectangle). cv2.detectMultiScale – функція для виявлення об'єктів на зображенні, використовуючи класифікатор каскаду та метод скейлінгу. Її параметри:

1. image – вхідне зображення, на якому Ви хочете виявити об'єкти.
2. scaleFactor – цей параметр визначає, наскільки зменшується розмір зображення під час кожного масштабування для більш швидкого пошуку об'єктів. Наприклад, якщо scaleFactor = 1.1, зображення буде зменшуватися на 10% під час кожного масштабування.
3. minNeighbors – мінімальна кількість сусідніх об'єктів, яка повинна бути в окрузі потенційного об'єкта, щоб вважати його коректним. Це допомагає відсікати помилкові виявлення.

4. minSize – мінімальний розмір об'єкта, який потрібно виявити. Це кортеж з двох значень (ширина, висота), і об'єкти меншого розміру будуть проігноровані.

Отримані від функції cv2.detectMultiScale числа є набором, для відображення прямокутників – перші два значення формують координати лівої верхньої точки, а наступні два є довжиною та шириною прямокутника відповідно. Для створення цих прямокутників слід використати функцію cv2.rectangle. Її параметри:

1. img – вхідне зображення, на якому Ви хочете намалювати прямокутник.
2. pt1 – координати верхнього лівого кута прямокутника у форматі (x, y).
3. pt2 – координати нижнього правого кута прямокутника у форматі (x, y).
4. color – колір прямокутника у форматі (B, G, R).
5. thickness – товщина лінії прямокутника. Якщо Ви вказуєте 0, то прямокутник буде заповнений кольором.

```
import cv2
# Читання зображення
img = cv2.imread('man.png')
# Читання каскадного класифікатора
cascade = cv2.CascadeClassifier('lbpcascade_frontalface.xml')
# Перетворення зображення на сіре
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Розпізнання обличчя
rects = cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=4, minSize=(40, 40), flags=cv2.CASCADE_SCALE_IMAGE)
# Розмиття зображення
gray = cv2.GaussianBlur(gray, (7, 7), 2.5)
# Детектування країв
edges = cv2.Canny(gray, 0, 40)
# Перетворення зображення назад у BGR
out = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)
# Навколо знайденого обличчя створюємо червоний прямокутник
for x, y, w, h in rects:
    cv2.rectangle(out, (x, y), (x + w, y + h), (0, 0, 255), 2)
# Відображення кінцевого зображення
cv2.imshow("edges+face", out)
# Очікування закриття вікна
cv2.waitKey(0)
```

Результат роботи коду (Рисунок 2.76).

У наведеному коді використано каскадний класифікатор, який розпізнає обличчя з фронтальної сторони, проте існує багато інших видів класифікаторів з відкритим доступом:

- детектор обличчя (за замовчуванням): haarcascade_frontalface_default.xml
- детектор обличчя (швидкий Xappi): haarcascade_frontalface_alt2.xml
- детектор обличчя (вид збоку): haarcascade_profileface.xml
- детектор очей (ліве око): haarcascade_lefteye_2splits.xml
- детектор очей (праве око): haarcascade_righteye_2splits.xml
- детектор рота: haarcascade_mcs_mouth.xml
- детектор носа: haarcascade_mcs_nose.xml
- детектор всього тіла: haarcascade_fullbody.xml
- детектор обличчя (швидкий LBP): lbpcascade_frontalface.xml

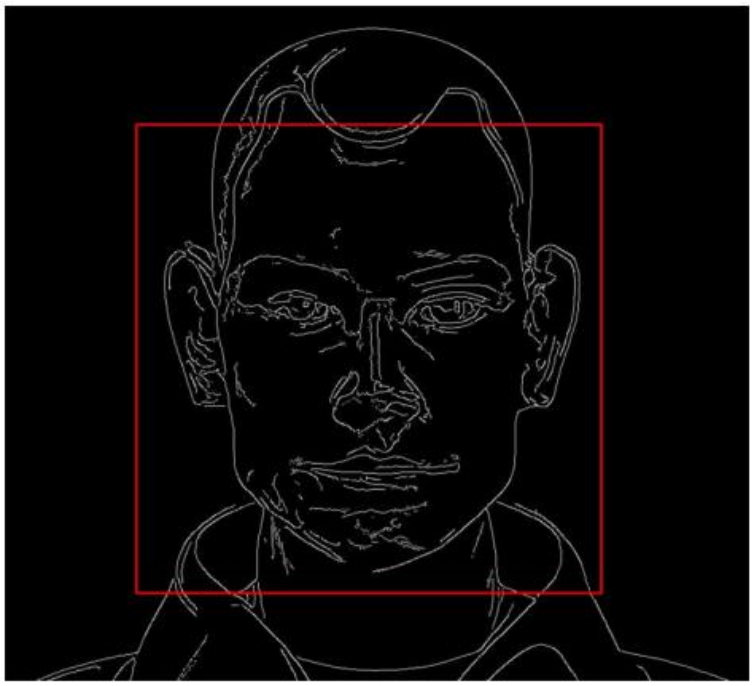


Рисунок 2.76 – Результат роботи коду із попереднього прикладу

2.5.7. Робота з відео у OpenCV

Робота з відеофайлами чи потоковим відео у пакеті OpenCV дуже подібна до обробки звичайних зображень, оскільки відео – це послідовний набір кадрів.

У коді робота з відео обов'язково пов'язана із об'єктами класу VideoCapture. За його допомоги можна працювати із відеофайлами або захоплювати відео із камери. У конструктор можна передати шлях до відеофайлу або ціле число, яке вказує на порядковий номер камери (можливі варіанти підключення кількох камер).

У об'єктів класу VideoCapture є метод read(), який повертає два об'єкти: логічне значення (True або False) та поточний кадр. Логічне значення відповідає за готовність кадру, а отриманий кард можна вивести на екран без попередньої обробки.

Приклад найпростішого коду для роботи із потоковим відео з камери:

```
import cv2

cap = cv2.VideoCapture(0)  # 0 - номер вбудованої камери

while cap.isOpened():  # Основний цикл
    # Зчитування кадру з камери
    ret, frame = cap.read()
    # Перевірка готовності кадру
    if not ret:
        break
    # Відображення відео покадрово
    cv2.imshow("Video", frame)
    # Коли буде натиснуто клавішу q, закінчиться відображення
відео
    if cv2.waitKey(1) == ord('q'):
        break

# Звільнення камери та закриття всіх вікон
cap.release()
cv2.destroyAllWindows()
```

Результатом роботи цього коду є послідовність кадрів із камери.

Приклад найпростішого коду для роботи із відеофайлами:

```
import cv2

cap = cv2.VideoCapture('man.mkv')  # Шлях до відеофайлу

while cap.isOpened():  # Основний цикл
    # Зчитування кадру з відеофайлу
    ret, frame = cap.read()
    # Перевірка готовності кадру
    if not ret:
        break
    # Відображення відео покадрово
    cv2.imshow("Video", frame)
    # Коли буде натиснуто клавішу q, закінчиться відображення
відео
    if cv2.waitKey(1) == ord('q'):
        break

# Закриття всіх вікон
cv2.destroyAllWindows()
```

Результатом роботи цього коду є послідовність кадрів із відеофайлу. При цьому відтворення звуку відсутнє, оскільки пакет OpenCV призначений для комп'ютерного бачення, а не для мультимедійних задач.

2.5.7.1. Запис відео

Запис відео можна робити двома способами: записувати кожен кадр у окреме зображення або одразу записувати кадри у відеофайл. Для першого способу використовується функція cv2.imwrite(filename, img), що була описана у роботі із зображеннями. Для другого способу використовується об'єкт класу VideoWriter(), який має такі параметри:

1. filename – шлях до файлу, у якому записуватиметься відео. Його розширення повинно відповідати наступному параметру.
2. fourcc – 4-символьний код кодека, який використовується для стиснення кадрів. Створюється як об'єкт класу cv2.VideoWriter_fourcc(*code), де code – рядок із 4 символів (наприклад, 'mp4v' для розширення mp4, 'XVID' – для avi).
3. fps – кількість кадрів за секунду. Часто використовують 24, 30 чи 60.
4. size – розмір кадрів відео, кортеж із ширини та висоти.
5. isColor – якщо не 0, то кодуватиметься кольорове зображення. Інакше відбуватиметься запис у відтінках сірого.

Приклад коду із записом відео:

```
import cv2

cap = cv2.VideoCapture(0)
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('output.mp4', fourcc, 30, (640, 480))

count = 0
while cap.isOpened():
```

```
    ret, frame = cap.read()
    if not ret:
        break
    cv2.imshow("Video", frame)
    # Запис окремого кадру як зображення
    cv2.imwrite(f'video/video_{count}.png', frame)
    count += 1
    # Запис кадру у відеофайл
    out.write(frame)
    if cv2.waitKey(1) == ord('q'):
        break

cv2.destroyAllWindows()
```

Результатом роботи цього коду є запис окремих кадрів із камери у папку video та відеофайлу з розширенням mp4.

2.5.7.2. Складні операції із відео. Приклад розпізнавання обличь

У контексті пакету OpenCV, важливо відзначити, що відео можна розглядати як послідовність кадрів, і це відкриває безмежні можливості для обробки відеоданих. Коли ми працюємо з відео, основна логіка обробки зазвичай вкладена в цикл while, де кожен ітераційний крок представляє один кадр з відео. Це надає розробникам можливість застосовувати ті самі або подібні методи обробки, які вони використовують для зображень.

Такий підхід дозволяє реалізувати різноманітні завдання, такі як виявлення об'єктів на відео, відстеження руху, аналіз кольору та багато інших. Під час обробки кожного кадру можна застосовувати фільтри, виправляти спотворення та виконувати інші операції, які допомагають отримати більш деталізовану інформацію з відео.

Приклад розпізнавання обличчя у відео із камери (аналогічно до розділу 2.6.6.6):

```
import cv2

cap = cv2.VideoCapture(0)  # 0 - вбудована камера
# Читання каскадного класифікатора
cascade = cv2.CascadeClassifier('lbpcascade_frontalface.xml')

while cap.isOpened():
    # Зчитування кадру з камери
    ret, img = cap.read()
    # Перевірка готовності кадру
    if not ret:
        break
    # Перетворення зображення на сіре
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Розпізнання обличчя
    rects = cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=4, minSize=(40, 40), flags=cv2.CASCADE_SCALE_IMAGE)
    # Розмиття зображення
    gray = cv2.GaussianBlur(gray, (7, 7), 2.5)
    # Детектування країв
    edges = cv2.Canny(gray, 0, 40)
    # Перетворення зображення назад у BGR
    out = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)
```

```
    # Навколо знайденого обличчя створюємо червоний прямокутник
    for x, y, w, h in rects:
        cv2.rectangle(out, (x, y), (x + w, y + h), (0, 0, 255), 2)
    cv2.imshow('edges+face', out)
    if cv2.waitKey(30) == ord('q'):
        break

# Звільнення камери та закриття всіх вікон
cap.release()
cv2.destroyAllWindows()
```

Результатом роботи коду є відображення відео з розпізнаними краями та червоними квадратами навколо обличь.

Завдання до підрозділу «Пакет OpenCV та комп'ютерний зір»

1. Напишіть код, який завантажує із файла та показує зображення, яке зберігається на диску Вашого комп'ютера.
2. Допишіть в коді із попереднього завдання команди, які спочатку вирізають частину зображення, а потім виводять її на екран.
3. Додайте до коду команди, які спочатку обертають зображення навколо його центральної точки, змінюють його масштаб, а потім виводять його на екран.
4. Допишіть до коду команди, які малюють прямокутник червоного кольору на початковому зображенні.
5. Збережіть отримане в попередньому завданні зображення у файлі на диску Вашого комп'ютера.
6. Допишіть до коду команди, які перетворюють зображення у відтінки сірого кольору.

РОЗДІЛ 3. Пакет Biopython

Вебсайт Biopython http://www.biopython.org [44] надає онлайн-ресурси для модулів, скриптів та вебпосилань для розробників програмного забезпечення на основі python для використання та дослідження в області біоінформатики [45]. Мета Biopython – максимально спростити використання мови python для біоінформатики шляхом створення високоякісних модулів та класів, які можна використовувати багаторазово.

Функції Biopython включають:

- синтаксичні аналізатори для різних форматів файлів біоінформатики (BLAST, Clustalw, FASTA, ...)
- доступ до онлайн-сервісів (NCBI, Expasy, ...)
- інтерфейси до різних програм (BLAST, Clustalw, EMBOSS, …)
- стандартний клас послідовностей
- різні модулі кластеризації
- документацію тощо.

EMBOSS – безкоштовний пакет аналізу ПЗ з відкритим кодом, розроблений для потреб спільноти користувачів молекулярної біології та біоінформатики містить різноманітні програми для вирівнювання послідовностей, швидкого пошуку в базі даних із шаблонами послідовностей, ідентифікації білкових мотивів (включаючи аналіз доменів) тощо.

EMBOSS також об'єднує цілу низку доступних на даний момент пакетів та інструментів для аналізу послідовностей у єдине ціле.

Програмне забезпечення EMBOSS створене учасниками EMBnet (European Molecular Biology network). EMBnet представляє широку групу користувачів і тісно співпрацює з виробниками баз даних, такими як Європейський інститут біоінформатики (EMBL), Швейцарський інститут біоінформатики (Swiss-Prot), Мюнхенський інформаційний центр білкових послідовностей (MIPS) тощо.

Biopython має такі функціональні можливості:

- Biopython включає стандартні класи послідовностей, ідентифікатори послідовностей та функції для роботи з послідовностями.
- Інструменти для виконання загальних операцій над послідовностями, таких як трансляція, транскрипція та обчислення ваги.
- Код для роботи з вирівнюваннями, включаючи стандартний спосіб створення та роботи з матрицями замін.
- Код, що полегшує поділ паралельних задач на окремі процеси.
- Код для виконання класифікації даних за допомогою k найближчих сусідів, наївний байєсів класифікатор або підтримка векторних машин.
- Значна кількість документації та допомога з використанням модулів, документація з он-лайн вікі, вебсайтів та списків розсилки.
- Інтеграція з BioSQL, що є схемою бази даних послідовностей.
- Biopython також підтримується проєктами BioPerl та BioJava.

BioSQL – це загальна схема бази даних, призначена головним чином для зберігання послідовностей та пов'язаних із ними даних для всього ядра СУБД.

BioSQL спроєктований в такий спосіб, що у ньому зберігаються дані з усіх популярних баз даних з біоінформатики, як-от GenBank, Swissport, EMBL тощо.

3.1. Клас Seq модуля Bio.Seq

Клас Seq, визначений в модулі Bio.Seq. Модуль Bio.Seq використовується для маніпулювання даними послідовності, а клас Seq використовується для представлення даних конкретної заданої послідовності в коді або послідовності з файлу. Кожен екземпляр класу Seq має два атрибута:

- дані – фактична послідовність рядків (GATC)
- алфавіт (Alphabet) – використовується для представлення типу послідовності, наприклад, послідовність ДНК, послідовність РНК тощо. За замовчуванням цей атрибут не представляє будь-яку послідовність і носить загальний характер.

В наступному прикладі ми створюємо екземпляр класу Seq та виводимо його на друк:

```
from Bio.Seq import Seq
simple_seq = Seq("GATC")
print(f'simple_seq = {simple_seq}')

Результат роботи коду:
simple_seq = GATC
```

3.2. Клас SeqRecord модуля Bio.SeqRecord

Клас SeqRecord (Sequence Record/Запис послідовності) визначений в модулі Bio.SeqRecord. Цей модуль використовується для управління записами послідовності, а клас SeqRecord використовується для представлення певної послідовності та інформації про неї через атрибути класу.

Основні атрибути класу SeqRecord:

- .seq – сама послідовність, як правило, екземпляр (об'єкт) класу Seq.
- .id – первинний ідентифікатор, що використовується для ідентифікації послідовності-рядку. У більшості випадків це щось на кшталт унікального номера в БД «accession number».

Додаткові атрибути:

- .name – назва послідовності, наприклад, назва гену (рядок). У деяких випадках це буде те ж саме, що і «accession number». Це аналог ідентифікатора послідовності у записі GenBank.
- .description – додатковий текст (рядок), опис послідовності.
- .annotations – додаткова інформація про всю послідовність (словник). Ключі – це назва інформації, а інформація міститься у значенні. Це дозволяє додати до послідовності більше "неструктурованої" інформації. Більшість записів – це рядки або списки рядків.
- .features – перелік особливостей об’єктів SeqFeature, тобто більш структурована інформація про особливості послідовності (наприклад, розташування генів у геномі або доменів у білковій послідовності).
- .dbxrefs – список перехресних посилань на базу даних (список рядків) (список рядків) функції - будь-які визначені (під)функції (список об'єктів SeqFeature).
- .letter_annotations – зберігає анотації на літери послідовності за допомогою словника додаткової інформації про літери в послідовності. Ключі – це назва інформації, а інформація міститься у значенні словника про послідовність python (тобто список, кортеж або рядок) з такою ж довжиною, що і сама послідовність. Типовим використанням є зберігання списку цілих чисел, що представляють показники якості послідовності; рядка, що представляє інформацію про вторинну структуру білка.

Клас SeqRecord визначений в модулі Bio.SeqRecord використовується для представлення певної послідовності та інформації про неї, наприклад:

```
# Створення екземпляру simple_seq класу Seq
# та simple_seq1, simple_seq2 класу SeqRecord

from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
simple_seq = Seq("GATC")
simple_seq1 = SeqRecord(simple_seq)
simple_seq2 = SeqRecord("GATC")
```

```
print(f'simple_seq  Seq = {simple_seq}')
print(f'simple_seq1 SeqRecord = {simple_seq1}')
print(f'simple_seq2 SeqRecord = {simple_seq2.seq}')
```

```
Результат роботи коду:
simple_seq = GATC
simple_seq1 = ID: <unknown id>
Name: <unknown name>
Description: <unknown description>
Number of features: 0
Seq('GATC')
simple_seq2 SeqRecord = GATC
```

Ви також можете визначити ідентифікатор, ім'я та опис функції ініціалізації, але якщо ні, вони будуть встановлені як рядки, що вказують на те, що ці атрибути є невідомими, як у коді вище, і можуть бути змінені згодом.

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
simple_seq = Seq("GATC")
simple_seq = SeqRecord(simple_seq)
simple_seq.id = "AC12345"    # Завдання атрибутів послідовності
simple_seq.description = " Послідовність GATC"
print(simple_seq.description, simple_seq.id, '\n', simple_seq)
```

```
Результат роботи коду:
Послідовність GATC AC12345
ID: AC12345
Name: <unknown name>
Description: Послідовність GATC
Number of features: 0
Seq('GATC')
```

Завдання атрибутів є важливим, якщо Ви хочете записати послідовність у файл. Ви можете задаєте атрибути під час створення екземпляру класу SeqRecord:

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
simple_seq = Seq("GATCGATCTCATGATCTC")
simple_seq = SeqRecord(simple_seq)
simple_seq_a = SeqRecord(simple_seq, id="AC12345", name="ген xxx",
description="включає "
    "два повтори", annotations={'1:100': 'ген mamO', '120:200':
'ген mamP'})
print(simple_seq_a)
```

```
Результат роботи коду:
ID: AC12345
Name: ген xxx
Description: включає два повтори
Number of features: 0
/1:100=ген mamO
/120:200=ген mamP
```

Атрибут .letter_annotations – це словник, що дозволяє призначити будь-якій послідовності python рядок, список або кортеж, який має таку ж довжину як і послідовність. В даному прикладі .letter_annotations зберігає список цілих чисел, що представляють показники якості послідовності: "phred_quality " – це ключ, список [40, 40, 38, 30] – значення; довжина списку (4) дорівнює довжині послідовності GATC (4). В наступному прикладі використовується атрибут .letter_annotations:

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
simple_seq = Seq("GATC")
simple_seq = SeqRecord(simple_seq, id = "AC12345")
simple_seq.letter_annotations["phred_quality"] = [40, 40, 38, 30]
print(simple_seq.letter_annotations)
print(simple_seq.letter_annotations["phred_quality"])

Результат роботи коду:
{'phred_quality': [40, 40, 38, 30]}
[40, 40, 38, 30]
```

3.3. Використання Biopython для роботи з послідовностями БД

Biopython має такі можливості для роботи з послідовностями БД:

- Можна скопіювати послідовність безпосередньо з NCBI і вставити її в код. Але тоді у такої послідовності не будуть визначені ідентифікатори і характеристики, пов'язані з послідовністю.
- Можна завантажити файл з інформацією про послідовність у .fasta форматі з NCBI, зберегти його і потім із коду прочитати з цього файлу інформацію про послідовність.
- Можна безпосередньо із коду скачати інформацію про послідовність з NCBI.

Здійснимо копіювання послідовності безпосередньо з NCBI і вставлення її в код. Для цього на сайті https://www.ncbi.nlm.nih.gov/ [46] знаходимо послідовність, наприклад AY508230.1 (Рисунок 3.1).

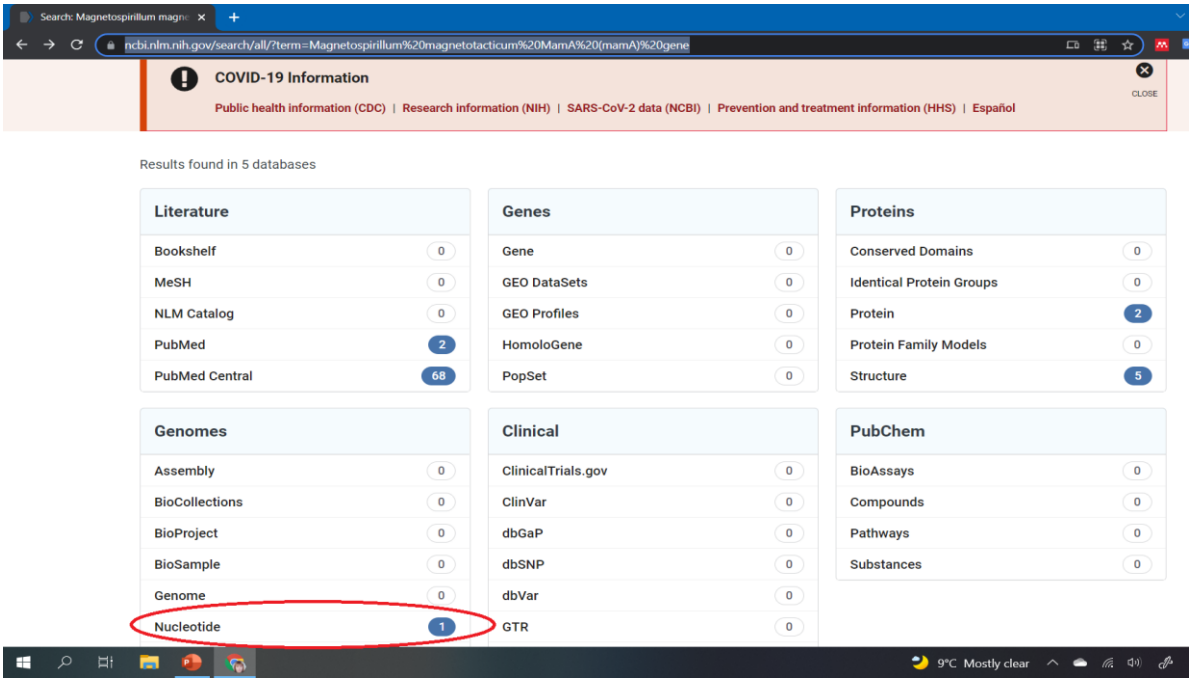


Рисунок 3.1 – Схема вибору генетичної послідовності на сайті https://www.ncbi.nlm.nih.gov/

Після здійснення попереднього кроку, зображеного на Рисунку 3.1 червоним контуром на сайті https://www.ncbi.nlm.nih.gov/ відображається послідовність AY508230.1 (Magnetospirillum magnetotacticum MamA) з нумерацією рядків (Рисунок 3.2):

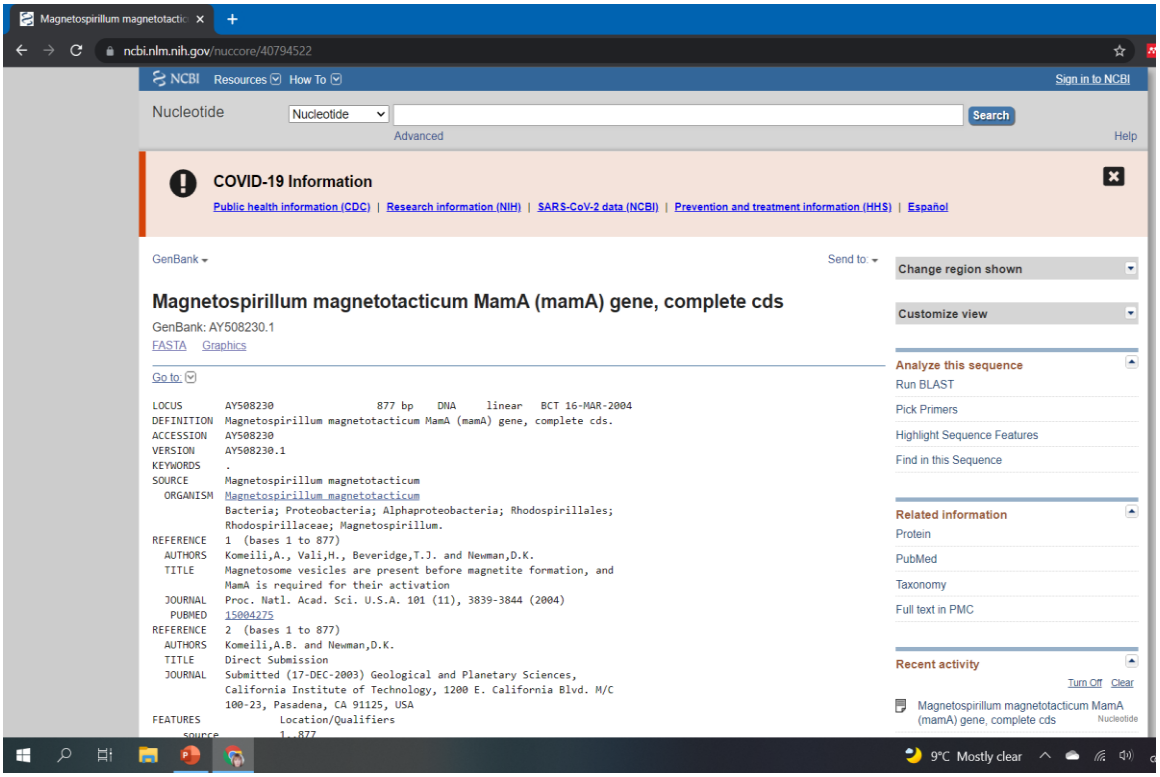


Рисунок 3.2 – Послідовність AY508230.1 (Magnetospirillum magnetotacticum MamA) з нумерацією рядків.

Для копіювання послідовності безпосередньо з NCBI і вставлення її в код вибираємо формат FASTA, в якому номери рядків послідовності не відображаються (Рисунок 3.3).

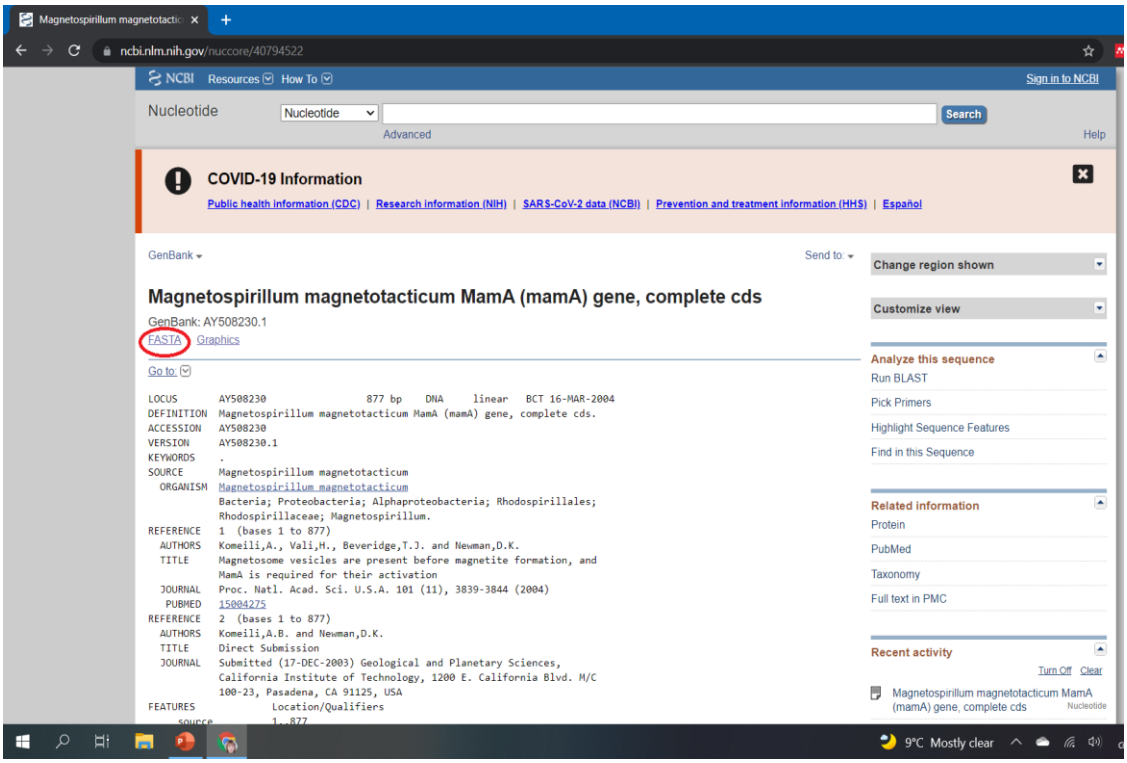


Рисунок 3.3 – Вибір формату FASTA.

Після переходу на сторінку з FASTA форматом послідовності, копіюємо послідовність в код при створенні екземпляру класу Seq (Рисунок 3.4).

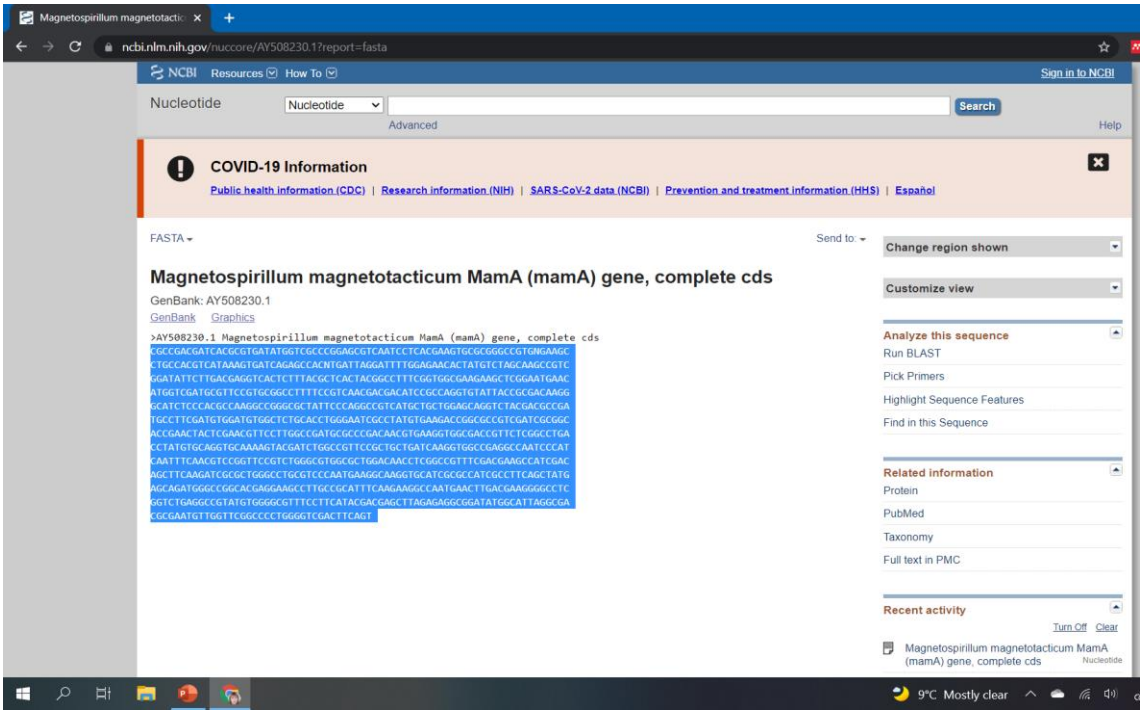


Рисунок 3.4 – Копіювання послідовності в код при створенні екземпляру класу Seq.

Безперечно центральним об'єктом біоінформатики є послідовність. Таким чином, ми продовжимо ознайомлення з методами Biopython для роботи з послідовностями, об’єктом Seq, в даному прикладі – це ген mamA магнітотаксисної бактерії Magnetospirillum magnetotacticum:

```
# Створюємо екземпляр my_seq класу Seq
from Bio.Seq import Seq
my_seq =
Seq('CGCCGACGATCACGCGTGATATGGTCGCCCGGAGCGTCAATCCTCACGAAGTGCGCGGGCC
GTGNGAAGC'
'CTGCCACGTCATAAAGTGATCAGAGCCACNTGATTAGGATTTTGGAGAACACTATGTCTAGCAAG
CCGTC'
'GGATATTCTTGACGAGGTCACTCTTTACGCTCACTACGGCCTTTCGGTGGCGAAGAAGCTCGGAA
TGAAC'
'ATGGTCGATGCGTTCCGTGCGGCCTTTTCCGTCAACGACGACATCCGCCAGGTGTATTACCGCGA
CAAGG'
'GCATCTCCCACGCCAAGGCCGGGCGCTATTCCCAGGCCGTCATGCTGCTGGAGCAGGTCTACGAC
GCCGA'
'TGCCTTCGATGTGGATGTGGCTCTGCACCTGGGAATCGCCTATGTGAAGACCGGCGCCGTCGATC
GCGGC'
'ACCGAACTACTCGAACGTTCCTTGGCCGATGCGCCCGACAACGTGAAGGTGGCGACCGTTCTCGG
CCTGA'
'CCTATGTGCAGGTGCAAAAGTACGATCTGGCCGTTCCGCTGCTGATCAAGGTGGCCGAGGCCAAT
CCCAT'
'CAATTTCAACGTCCGGTTCCGTCTGGGCGTGGCGCTGGACAACCTCGGCCGTTTCGACGAAGCCA
TCGAC'
'AGCTTCAAGATCGCGCTGGGCCTGCGTCCCAATGAAGGCAAGGTGCATCGCGCCATCGCCTTCAG
CTATG'
```

```
'AGCAGATGGGCCGGCACGAGGAAGCCTTGCCGCATTTCAAGAAGGCCAATGAACTTGACGAAGGG
GCCTC'
'GGTCTGAGGCCGTATGTGGGGCGTTTCCTTCATACGACGAGCTTAGAGAGGCGGATATGGCATTA
GGCGA'
'CGCGAATGTTGGTTCGGCCCCTGGGGTCGACTTCAG')
Comment = 'Magnetospirillum magnetotacticum MamA (mamA) gene,
complete cds GenBank: AY508230.1'
print(my_seq)
```

Результат роботи коду:

```
CGCCGACGATCACGCGTGATATGGTCGCCCGGAGCGTCAATCCTCACGAAGTGCGCGGGCCGTGNG
AAGCCTGCCACGTCATAAAGTGATCAGAGCCACNTGATTAGGATTTTGGAGAACACTATGTCTAGC
AAGCCGTCGGATATTCTTGACGAGGTCACTCTTTACGCTCACTACGGCCTTTCGGTGGCGAAGAAG
CTCGGAATGAACATGGTCGATGCGTTCCGTGCGGCCTTTTCCGTCAACGACGACATCCGCCAGGTG
TATTACCGCGACAAGGGCATCTCCCACGCCAAGGCCGGGCGCTATTCCCAGGCCGTCATGCTGCTG
GAGCAGGTCTACGACGCCGATGCCTTCGATGTGGATGTGGCTCTGCACCTGGGAATCGCCTATGTG
AAGACCGGCGCCGTCGATCGCGGCACCGAACTACTCGAACGTTCCTTGGCCGATGCGCCCGACAAC
GTGAAGGTGGCGACCGTTCTCGGCCTGACCTATGTGCAGGTGCAAAAGTACGATCTGGCCGTTCCG
CTGCTGATCAAGGTGGCCGAGGCCAATCCCATCAATTTCAACGTCCGGTTCCGTCTGGGCGTGGCG
CTGGACAACCTCGGCCGTTTCGACGAAGCCATCGACAGCTTCAAGATCGCGCTGGGCCTGCGTCCC
AATGAAGGCAAGGTGCATCGCGCCATCGCCTTCAGCTATGAGCAGATGGGCCGGCACGAGGAAGCC
TTGCCGCATTTCAAGAAGGCCAATGAACTTGACGAAGGGGCCTCGGTCTGAGGCCGTATGTGGGGC
GTTTCCTTCATACGACGAGCTTAGAGAGGCGGATATGGCATTAGGCGACGCGAATGTTGGTTCGGC
CCCTGGGGTCGACTTCAG
```

Для завантаження і збереження файлу з інформацією про послідовність у .fasta форматі з NCBI Вибираємо send to (Рисунок 3.5).

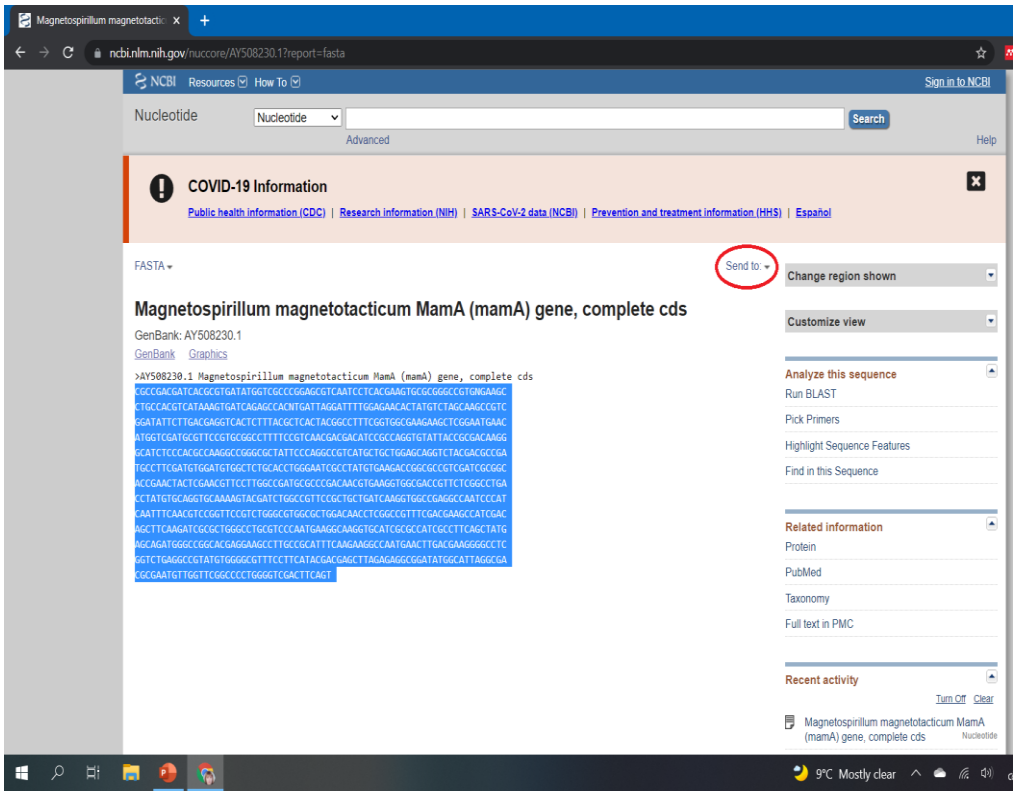


Рисунок 3.5 – Завантаження та збереження файлу з інформацією про послідовність у .fasta форматі з NCBI.

Після натискання send to вибираємо (Рисунок 3.6):
Complete Record
File
Format – FASTA
Show GI – Create File

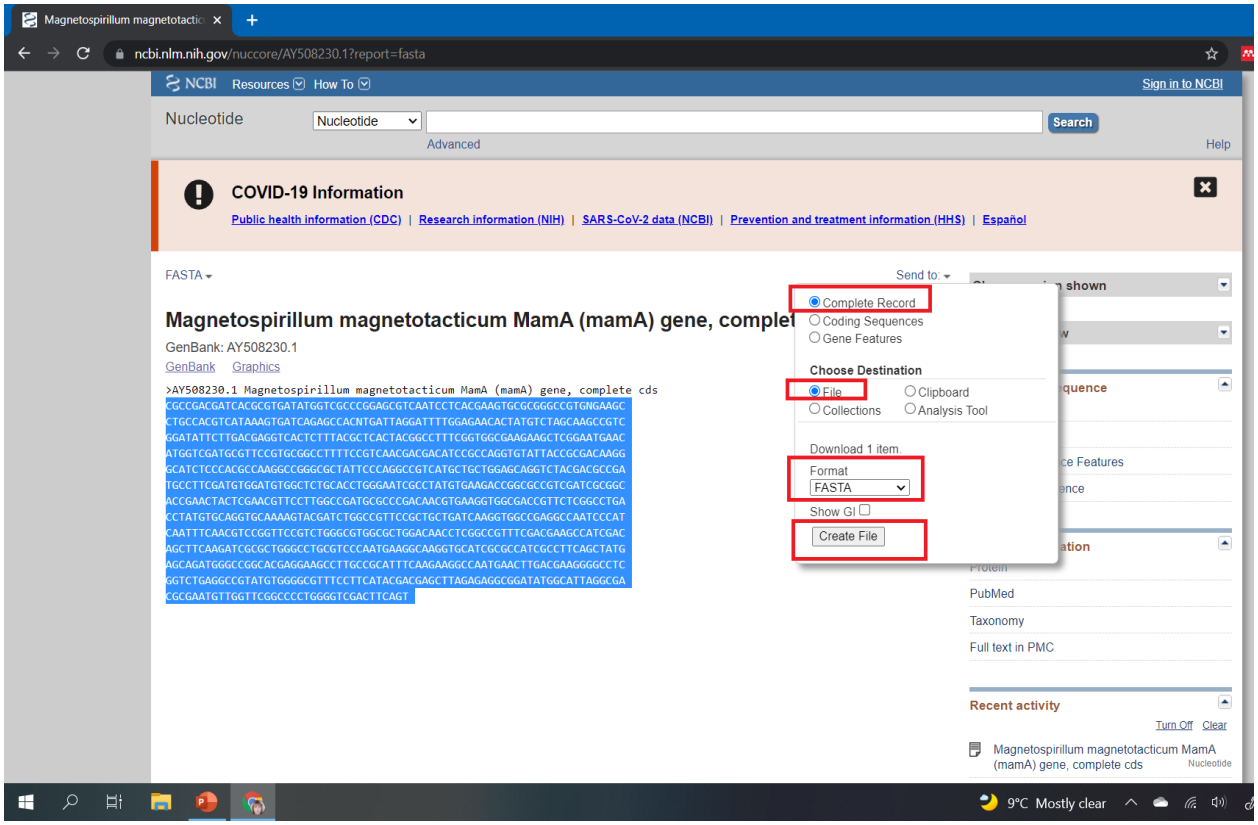


Рисунок 3.6 – Вибір формату файлу для збереження.

Завантажуємо автоматично створений файл sequence.fasta і зберігаємо у папці, яку створюємо всередині папки, в якій зберігається наш код (Рисунок 3.7).

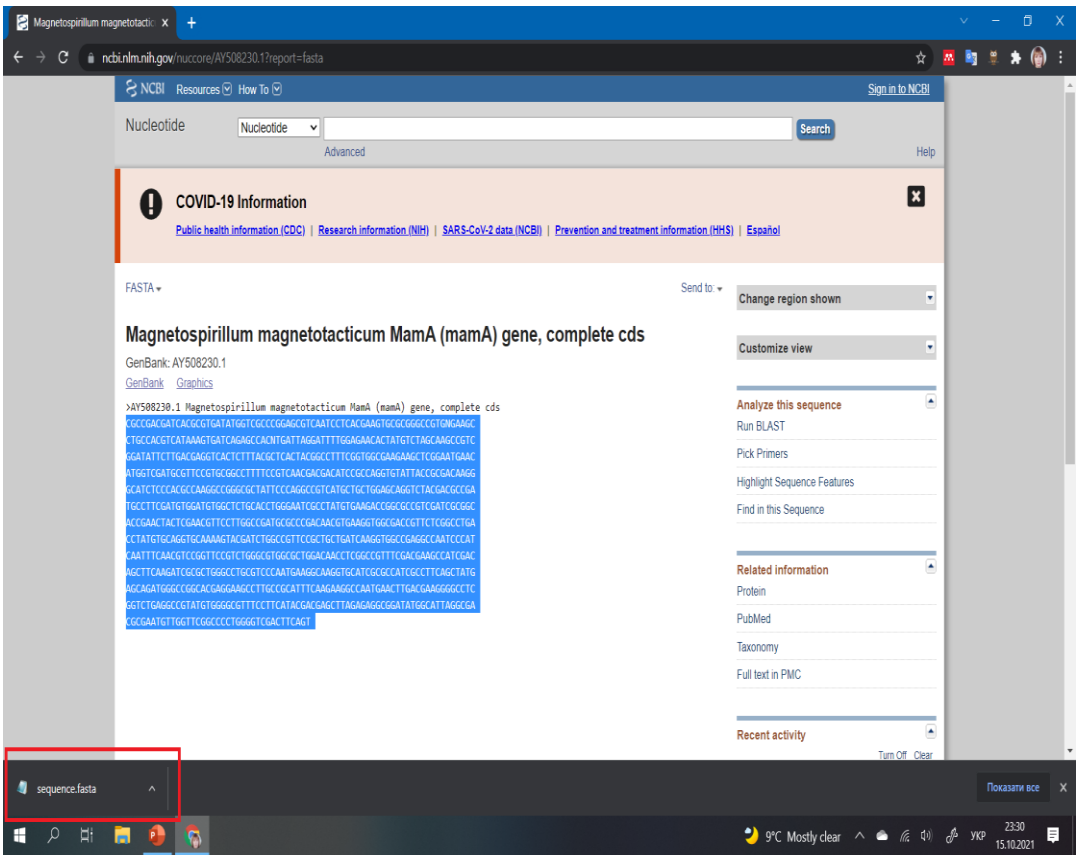


Рисунок 3.7 – Завантаження та збереження автоматично створеного файлу sequence.fasta.

Якщо не вдається описаним методом знайти інформацію про нуклеотид, наприклад, білка MamB Magnetospirillum gryphiswaldense MSR-1, використовуємо інший спосіб. Заносимо назву білка і мікроорганізму (MamB Magnetospirillum gryphiswaldense MSR-1) в пошуковий рядок NCBI, натискаємо Search, отримаємо інформацію, представлену на Рисунку 3.8.

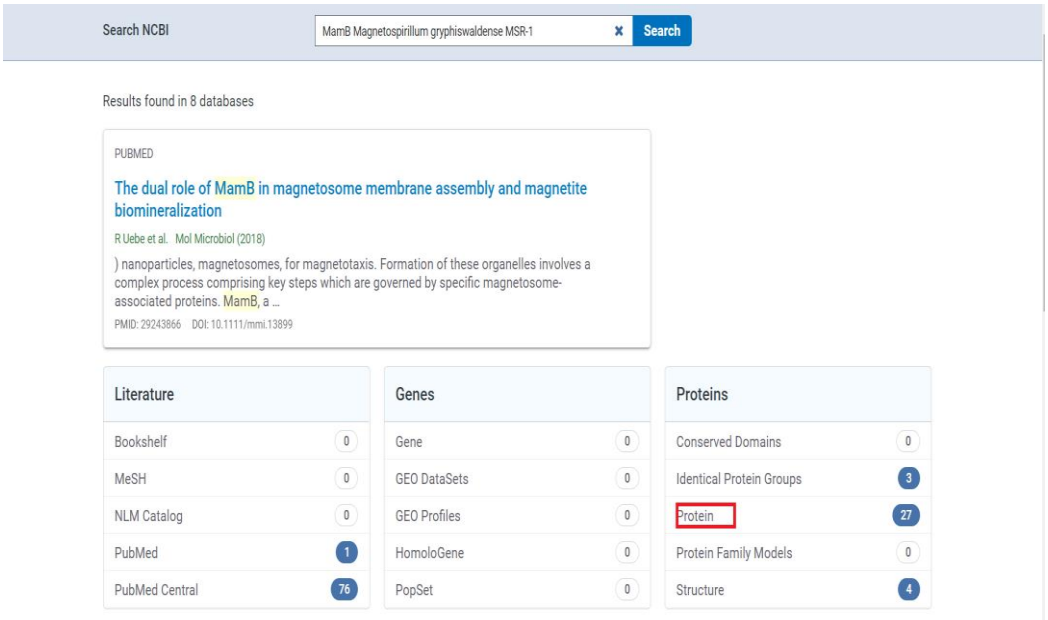


Рисунок 3.8 – Пошук назви білка і мікроорганізму (MamB Magnetospirillum gryphiswaldense MSR-1) в пошуковому рядку NCBI.

Вибираємо Protein (Рисунок 3.8).

На сторінці NCBI, що відкрилася, вибираємо досліджуваний білок magnetosome protein MamB [Magnetospirillum gryphiswaldense MSR-1] (Рисунок 3.9).

Рисунок 3.9 – Вибір досліджуваного білка magnetosome protein MamB [Magnetospirillum gryphiswaldense MSR-1].

На сторінці NCBI, що відкрилася, вибираємо Nucleotide. (Рисунок 3.10)

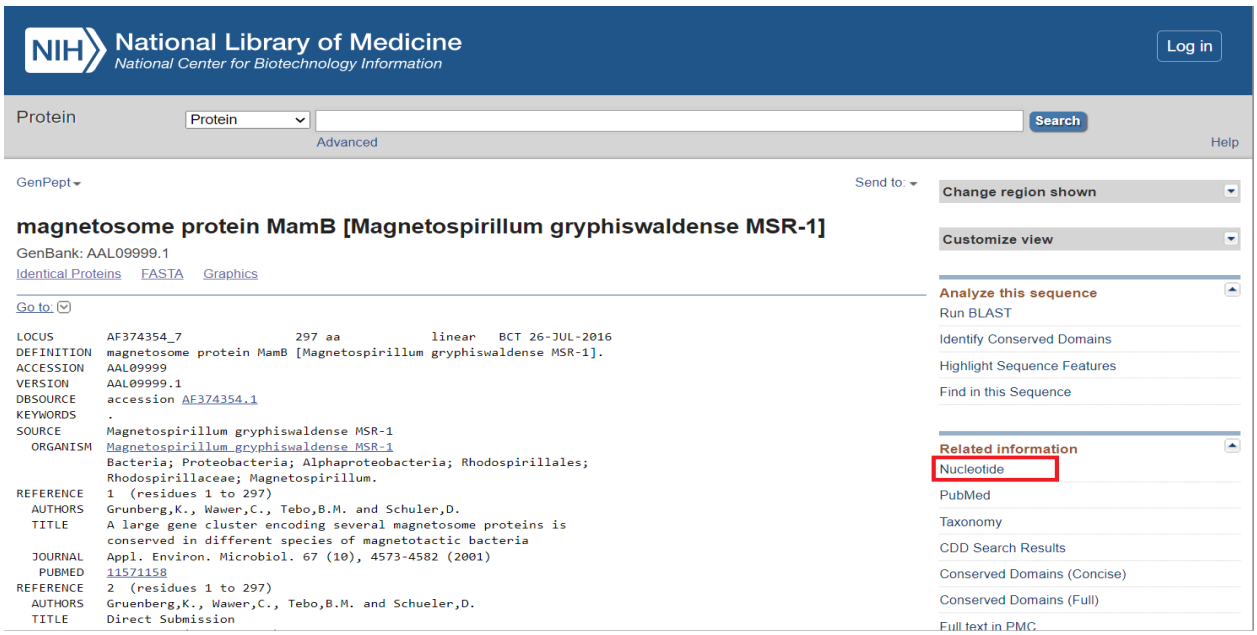


Рисунок 3.10 – Вибір досліджуваного білка magnetosome protein MamB [Magnetospirillum gryphiswaldense MSR-1].

На новій сторінці шукаємо таку інформацію:

gene 5085..5978

/gene="mamB" CDS 5085..5978

Та вибираємо CDS.

На наступній сторінці NCBI, що відкрилася, вибираємо: GenBank (Рисунок 3.11).

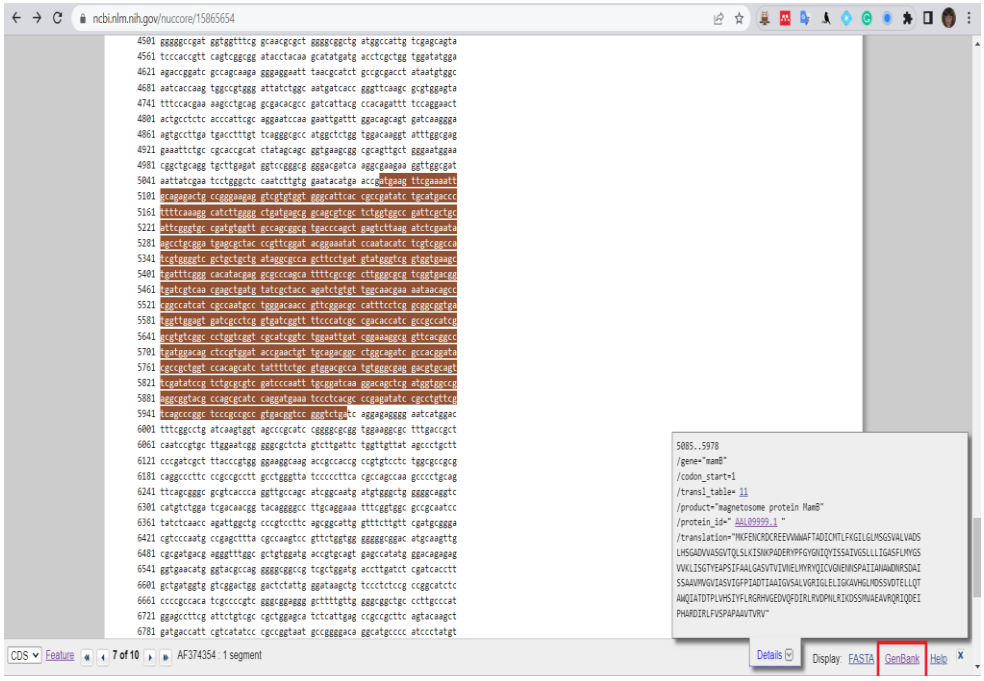


Рисунок 3.11 – Завантаження і збереження файлу з інформацією про послідовність у .fasta форматі з NCBI.

На наступній сторінці NCBI, що відкрилася, вибираємо: FASTA, повторюємо дії описані вище для попереднього прикладу. В результаті зберігаємо файл, наприклад, sequence(47).fasta папці, яку створюємо всередині папки, в якій зберігається наш код (Рисунок 3.12).

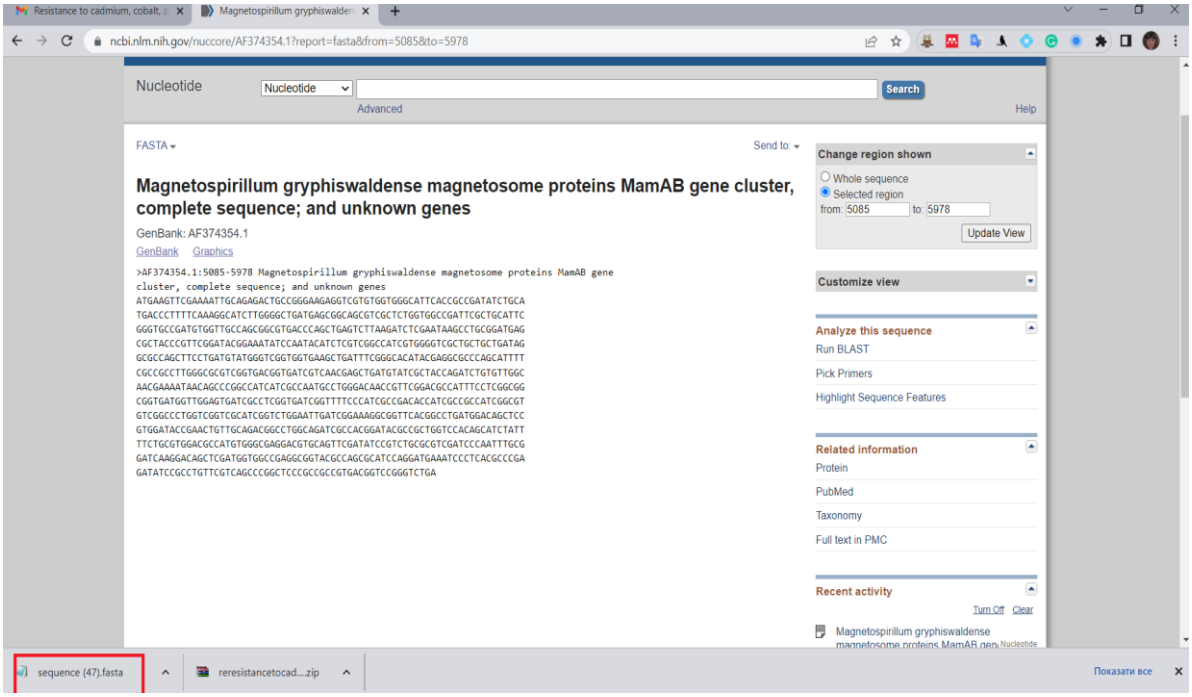


Рисунок 3.12 – Завантаження і збереження файлу з інформацією про послідовність у .fasta форматі з NCBI.

Розглянемо зчитування послідовності з файлу, який попередньо завантажено з NCBI. Аргументами функції parse() модуля SeqIO є ім'я файлу та формат файлу, повертає функція parse() ітерований об’єкт класу SeqRecord, який при друку самого об’єкта, наприклад seq_record, друкує всі атрибути класу SeqRecord. Наприклад:

```
from Bio import SeqIO
for seq_record in SeqIO.parse("BioSeqExamples\sequenceACGT.fasta",
"fasta"):
    print(f'seq_record.id = {seq_record.id}')
    print(f'repr(seq_record.seq) = {repr(seq_record.seq)}')
    print(f'len(seq_record) = {len(seq_record)}')
```

```
Результат роботи коду:
seq_record.id = AY508230.1
repr(seq_record.seq)=  Seq('CGCCGACGATCACGGGCGTCAATCCAGTGC...AGT')
len(seq_record) = 877
```

Наступний приклад показує зчитування послідовності з файлу попередньо завантаженого з NCBI з використанням функцій open та parse:

```
from Bio import SeqIO
with open(r"BioSeqExamples\sequenceAGCT.fasta") as fasta_file:
    for seq_record in SeqIO.parse(fasta_file, "fasta"):
        print("ID:", seq_record.id)
        print("Name:", seq_record.name)
        print("Description:", seq_record.description)
        print("Annotations:", seq_record.annotations)
        print("Sequence features:", seq_record.features)
        print("Sequence Data:", seq_record.seq)
```

```
Результат роботи коду:
ID: AY508230.1
Name: AY508230.1
Description: AY508230.1 Magnetospirillum magnetotacticum MamA
(mamA) gene, complete cds
Annotations: {}
Sequence features: []
Sequence Data:
CGCCGACGATCACGCGTGATATGGTCGCCCGGAGCGTCAATCCTCACGAAGTGCGCGGGCCGTGNG
AAGCCTGCCACGTCATAAAGTGATCAGAGCCACNTGATTAGGATTTTGGAGAACACTATGTCTAGC
AAGCCGTCGGATATTCTTGACGAGGTCACTCTTTACGCTCACTACGGCCTTTCGGTGGCGAAGAAG
CTCGGAATGAACATGGTCGATGCGTTCCGTGCGGCCTTTTCCGTCAACGACGACATCCGCCAGGTG
TATTACCGCGACAAGGGCATCTCCCACGCCAAGGCCGGGCGCTATTCCCAGGCCGTCATGCTGCTG
GAGCAGGTCTACGACGCCGATGCCTTCGATGTGGATGTGGCTCTGCACCTGGGAATCGCCTATGTG
AAGACCGGCGCCGTCGATCGCGGCACCGAACTACTCGAACGTTCCTTGGCCGATGCGCCCGACAAC
GTGAAGGTGGCGACCGTTCTCGGCCTGACCTATGTGCAGGTGCAAAAGTACGATCTGGCCGTTCCG
CTGCTGATCAAGGTGGCCGAGGCCAATCCCATCAATTTCAACGTCCGGTTCCGTCTGGGCGTGGCG
CTGGACAACCTCGGCCGTTTCGACGAAGCCATCGACAGCTTCAAGATCGCGCTGGGCCTGCGTCCC
AATGAAGGCAAGGTGCATCGCGCCATCGCCTTCAGCTATGAGCAGATGGGCCGGCACGAGGAAGCC
TTGCCGCATTTCAAGAAGGCCAATGAACTTGACGAAGGGGCCTCGGTCTGAGGCCGTATGTGGGGC
GTTTCCTTCATACGACGAGCTTAGAGAGGCGGATATGGCATTAGGCGACGCGAATGTTGGTTCGGC
CCCTGGGGTCGACTTCAGT
```

3.4. Основні операції над послідовностями в класі Seq модуля Bio.Seq

Ми продовжуємо ознайомлення з методами Biopython для роботи з послідовностями, об’єктом Seq, в даному прикладі – це ген mamA магнітотаксисної бактерії Magnetospirillum magnetotacticum:

```
# Зчитування об’єктів (послідовностей (послідовності) та їх
атрибутів з файлу, попередньо завантаженого з NCBI методом parse()
# Створюємо екземпляр my_seq класу Seq
from Bio.Seq import Seq
my_seq =
Seq('CGCCGACGATCACGCGTGATATGGTCGCCCGGAGCGTCAATCCTCACGAAGTGCGCGGGCC
GTGNGAAGC'
'CTGCCACGTCATAAAGTGATCAGAGCCACNTGATTAGGATTTTGGAGAACACTATGTCTAGCAAG
CCGTC'
'GGATATTCTTGACGAGGTCACTCTTTACGCTCACTACGGCCTTTCGGTGGCGAAGAAGCTCGGAA
TGAAC'
'ATGGTCGATGCGTTCCGTGCGGCCTTTTCCGTCAACGACGACATCCGCCAGGTGTATTACCGCGA
CAAGG'
'GCATCTCCCACGCCAAGGCCGGGCGCTATTCCCAGGCCGTCATGCTGCTGGAGCAGGTCTACGAC
GCCGA'
'TGCCTTCGATGTGGATGTGGCTCTGCACCTGGGAATCGCCTATGTGAAGACCGGCGCCGTCGATC
GCGGC'
'ACCGAACTACTCGAACGTTCCTTGGCCGATGCGCCCGACAACGTGAAGGTGGCGACCGTTCTCGG
CCTGA'
'CCTATGTGCAGGTGCAAAAGTACGATCTGGCCGTTCCGCTGCTGATCAAGGTGGCCGAGGCCAAT
CCCAT'
'CAATTTCAACGTCCGGTTCCGTCTGGGCGTGGCGCTGGACAACCTCGGCCGTTTCGACGAAGCCA
TCGAC'
'AGCTTCAAGATCGCGCTGGGCCTGCGTCCCAATGAAGGCAAGGTGCATCGCGCCATCGCCTTCAG
CTATG'
'AGCAGATGGGCCGGCACGAGGAAGCCTTGCCGCATTTCAAGAAGGCCAATGAACTTGACGAAGGG
GCCTC'
'GGTCTGAGGCCGTATGTGGGGCGTTTCCTTCATACGACGAGCTTAGAGAGGCGGATATGGCATTA
GGCGA'
'CGCGAATGTTGGTTCGGCCCCTGGGGTCGACTTCAG')
Comment = 'Magnetospirillum magnetotacticum MamA (mamA) gene,
complete cds GenBank: AY508230.1'
print(my_seq)
```

Ви також можете використовувати екземпляр my_seq класу Seq безпосередньо при форматуванні рядків послідовності:

```
# Форматування послідовності
fasta_format_string = '\n>AY508230.1\n'+ my_seq + '\n'
print(f'fasta_format_string = {fasta_format_string}')
```

```
Результат роботи коду:
fasta_format_string =
>AY508230.1
CGCCGACGATCACGCGTGATATGGTCGCCCGGAGCGTCAATCCTCACGAAGTGCGCGGGCCGTGNG
AAGCCTGCCACGTCATAAAGTGATCAGAGCCACNTGATTAGGATTTTGGAGAACACTATGTCTAGC
AAGCCGTCGGATATTCTTGACGAGGTCACTCTTTACGCTCACTACGGCCTTTCGGTGGCGAAGAAG
CTCGGAATGAACATGGTCGATGCGTTCCGTGCGGCCTTTTCCGTCAACGACGACATCCGCCAGGTG
TATTACCGCGACAAGGGCATCTCCCACGCCAAGGCCGGGCGCTATTCCCAGGCCGTCATGCTGCTG
GAGCAGGTCTACGACGCCGATGCCTTCGATGTGGATGTGGCTCTGCACCTGGGAATCGCCTATGTG
AAGACCGGCGCCGTCGATCGCGGCACCGAACTACTCGAACGTTCCTTGGCCGATGCGCCCGACAAC
GTGAAGGTGGCGACCGTTCTCGGCCTGACCTATGTGCAGGTGCAAAAGTACGATCTGGCCGTTCCG
CTGCTGATCAAGGTGGCCGAGGCCAATCCCATCAATTTCAACGTCCGGTTCCGTCTGGGCGTGGCG
CTGGACAACCTCGGCCGTTTCGACGAAGCCATCGACAGCTTCAAGATCGCGCTGGGCCTGCGTCCC
```

```
AATGAAGGCAAGGTGCATCGCGCCATCGCCTTCAGCTATGAGCAGATGGGCCGGCACGAGGAAGCC
TTGCCGCATTTCAAGAAGGCCAATGAACTTGACGAAGGGGCCTCGGTCTGAGGCCGTATGTGGGGC
GTTTCCTTCATACGACGAGCTTAGAGAGGCGGATATGGCATTAGGCGACGCGAATGTTGGTTCGGC
CCCTGGGGTCGACTTCAG
```

Розглянемо основні операції над послідовностями, що доступні в класі Seq. Послідовності схожі на рядки python. Ми можемо виконувати операції з послідовностями python, такі як слайси, підрахунок, об'єднання, пошук підпослідовностей, розбиття на підпослідовності.
Наступний приклад ілюструє роботу з окремими літерами послідовності:

```
# Робота з окремими літерами послідовності
print(f'my_seq[0] = {my_seq[0]}')                    # перша
літера
print(f'my_seq[2]) = {my_seq[2]}')                   # третя
літера
print(f'my_seq[-1] = {my_seq[-1]}')                  # остання
літера
```

```
Результат роботи коду:
my_seq[0] = C
my_seq[2]) = C
my_seq[-1] = T
```

Низка методів, що використовуються для роботи з рядками, використовуються також для генетичних послідовностей, що важливо, наприклад, для визначення базового складу ДНК. Базовий склад геномної ДНК (наприклад, вміст підпослідовності CG) може істотно впливати на функціонування геному. Наприклад, в наступному прикладі здійснюється підрахунок кількості та порядкового номеру підпослідовностей:

```
# Підрахунок кількості, порядкового номеру підпослідовностей  тощо
from Bio.Seq import Seq
n_count = my_seq.count("T")
n_count1 = my_seq.count("CG")
n_find = my_seq.find("AT")              # Знаходження індексу
підпослідовності AT
n_split = my_seq.split("AG")            # Розділення послідовності
по AG
print(f'n_count = {n_count, n_count1, n_find, n_split}')
```

```
Результат роботи коду:
181   91   8   [Seq(''), Seq('C'), Seq('A'), Seq('ATCA'), Seq(''),
Seq('TGATATGGT'),   Seq('CC'),   Seq('GAG'),   Seq('TCAATCCTCA'),
Seq('AAGTG'), Seq(''), ….
```

Підрахунок кількості конкретних підпослідовностей у генетичній послідовності важливо здійснювати, наприклад, для розв’язання задачі про так звані CpG острівці. Підпослідовність літер CG – це CpG острівці. Динуклеотиди CG представляють особливий інтерес з кількох причин. У соматичних клітинах ссавців та інших хребетних, метилювання цитозину відбувається майже повністю в CpG острівцях і є епігенетичним механізмом, необхідним для нормального розвитку. Нуклеотид C у CpG острівці дуже мінливий, причому переходи C до T (і комплементарні T до A) є найпоширенішими мутаціями. Вважається, що ≈30-кратне збільшення частоти мутацій для CpG пов’язане з ферментативним метилюванням

CpG з утворенням 5-метилцитозину (mC). Потім заміна mC призводить до посиленого мутагенезу. Швидше за все, з цієї причини частота CpG острівців в геномі ссавців в середньому приблизно в 5 разів нижча за очікувану на основі загальногеномного нуклеотидного складу. Розраховується частка підпослідовностей CG, кількість CG поділена на загальну кількість нуклеотидів.

Так само, як і з рядками у python можна робити слайси з початкового індексу кінцевого індексу і з завданням кроку (який за замовчуванням дорівнює 1), наприклад:

```
# Слайси в послідовностях, друкується вся послідовність, починаючи
з першого,
# другого і третього кодонів, тобто перевірка відкритої рамки
зчитування
my_seq0 = my_seq[0::]
print(f'my_seq0[0::] = {my_seq0}')
print('my_seq0.translate()', my_seq0.translate())
my_seq1 = my_seq[1::] + 'A'
print(f'my_seq1[1::] = {my_seq1}')
print('my_seq1.translate()', my_seq1.translate())
my_seq2 = my_seq[2::] + 'AA'
print(f'my_seq2[2::] = {my_seq2}')
print('my_seq2.translate()', my_seq2.translate())

Результат роботи коду:
my_seq0[0::] =
CGCCGACGATCACGCGTGATATGGTCGCCCGGAGCGTCAATCCTCACGAAGTGCGCGGGCCGTGNG
AAGCCTGCCACGTCATAAAGTGATCAGAGCCACNTGATTAGGATTTTGGAGAACACTATGTCTAGC
AAGCCGTCGGATATTCTTGACGAGGTCACTCTTTACGCTCACTACGGCCTTTCGGTGGCGAAGAAG
CTCGGAATGAACATGGTCGATGCGTTCCGTGCGGCCTTTTCCGTCAACGACGACATCCGCCAGGTG
TATTACCGCGACAAGGGCATCTCCCACGCCAAGGCCGGGCGCTATTCCCAGGCCGTCATGCTGCTG
GAGCAGGTCTACGACGCCGATGCCTTCGATGTGGATGTGGCTCTGCACCTGGGAATCGCCTATGTG
AAGACCGGCGCCGTCGATCGCGGCACCGAACTACTCGAACGTTCCTTGGCCGATGCGCCCGACAAC
GTGAAGGTGGCGACCGTTCTCGGCCTGACCTATGTGCAGGTGCAAAAGTACGATCTGGCCGTTCCG
CTGCTGATCAAGGTGGCCGAGGCCAATCCCATCAATTTCAACGTCCGGTTCCGTCTGGGCGTGGCG
CTGGACAACCTCGGCCGTTTCGACGAAGCCATCGACAGCTTCAAGATCGCGCTGGGCCTGCGTCCC
AATGAAGGCAAGGTGCATCGCGCCATCGCCTTCAGCTATGAGCAGATGGGCCGGCACGAGGAAGCC
TTGCCGCATTTCAAGAAGGCCAATGAACTTGACGAAGGGGCCTCGGTCTGAGGCCGTATGTGGGGC
GTTTCCTTCATACGACGAGCTTAGAGAGGCGGATATGGCATTAGGCGACGCGAATGTTGGTTCGGC
CCCTGGGGTCGACTTCAG
my_seq0.translate()
RRRSRVIWSPGASILTKCAGRXKPATS*SDQSHXIRILENTMSSKPSDILDEVTLYAHYGLSVAKK
LGMNMVDAFRAAFSVNDDIRQVYYRDKGISHAKAGRYSQAVMLLEQVYDADAFDVDVALHLGIAYV
KTGAVDRGTELLERSLADAPDNVKVATVLGLTYVQVQKYDLAVPLLIKVAEANPINFNVRFRLGVA
LDNLGRFDEAIDSFKIALGLRPNEGKVHRAIAFSYEQMGRHEEALPHFKKANELDEGASV*GRMWG
VSFIRRA*RGGYGIRRRECWFGPWGRLQ
my_seq1[1::] =
GCCGACGATCACGCGTGATATGGTCGCCCGGAGCGTCAATCCTCACGAAGTGCGCGGGCCGTGNGA
AGCCTGCCACGTCATAAAGTGATCAGAGCCACNTGATTAGGATTTTGGAGAACACTATGTCTAGCA
AGCCGTCGGATATTCTTGACGAGGTCACTCTTTACGCTCACTACGGCCTTTCGGTGGCGAAGAAGC
TCGGAATGAACATGGTCGATGCGTTCCGTGCGGCCTTTTCCGTCAACGACGACATCCGCCAGGTGT
ATTACCGCGACAAGGGCATCTCCCACGCCAAGGCCGGGCGCTATTCCCAGGCCGTCATGCTGCTGG
AGCAGGTCTACGACGCCGATGCCTTCGATGTGGATGTGGCTCTGCACCTGGGAATCGCCTATGTGA
AGACCGGCGCCGTCGATCGCGGCACCGAACTACTCGAACGTTCCTTGGCCGATGCGCCCGACAACG
TGAAGGTGGCGACCGTTCTCGGCCTGACCTATGTGCAGGTGCAAAAGTACGATCTGGCCGTTCCGC
```

```
TGCTGATCAAGGTGGCCGAGGCCAATCCCATCAATTTCAACGTCCGGTTCCGTCTGGGCGTGGCGC
TGGACAACCTCGGCCGTTTCGACGAAGCCATCGACAGCTTCAAGATCGCGCTGGGCCTGCGTCCCA
ATGAAGGCAAGGTGCATCGCGCCATCGCCTTCAGCTATGAGCAGATGGGCCGGCACGAGGAAGCCT
TGCCGCATTTCAAGAAGGCCAATGAACTTGACGAAGGGGCCTCGGTCTGAGGCCGTATGTGGGGCG
TTTCCTTCATACGACGAGCTTAGAGAGGCGGATATGGCATTAGGCGACGCGAATGTTGGTTCGGCC
CCTGGGGTCGACTTCAGA
my_seq1.translate()
ADDHA*YGRPERQSSRSARAVXSLPRHKVIRAT*LGFWRTLCLASRRIFLTRSLFTLTTAFRWRRS
SE*TWSMRSVRPFPSTTTSARCITATRASPTPRPGAIPRPSCCWSRSTTPMPSMWMWLCTWESPM*
RPAPSIAAPNYSNVPWPMRPTT*RWRPFSA*PMCRCKSTIWPFRC*SRWPRPIPSISTSGSVWAWR
WTTSAVSTKPSTASRSRWACVPMKARCIAPSPSAMSRWAGTRKPCRISRRPMNLTKGPRSEAVCGA
FPSYDELREADMALGDANVGSAPGVDFR
my_seq2[2::] =
CCGACGATCACGCGTGATATGGTCGCCCGGAGCGTCAATCCTCACGAAGTGCGCGGGCCGTGNGAA
GCCTGCCACGTCATAAAGTGATCAGAGCCACNTGATTAGGATTTTGGAGAACACTATGTCTAGCAA
GCCGTCGGATATTCTTGACGAGGTCACTCTTTACGCTCACTACGGCCTTTCGGTGGCGAAGAAGCT
CGGAATGAACATGGTCGATGCGTTCCGTGCGGCCTTTTCCGTCAACGACGACATCCGCCAGGTGTA
TTACCGCGACAAGGGCATCTCCCACGCCAAGGCCGGGCGCTATTCCCAGGCCGTCATGCTGCTGGA
GCAGGTCTACGACGCCGATGCCTTCGATGTGGATGTGGCTCTGCACCTGGGAATCGCCTATGTGAA
GACCGGCGCCGTCGATCGCGGCACCGAACTACTCGAACGTTCCTTGGCCGATGCGCCCGACAACGT
GAAGGTGGCGACCGTTCTCGGCCTGACCTATGTGCAGGTGCAAAAGTACGATCTGGCCGTTCCGCT
GCTGATCAAGGTGGCCGAGGCCAATCCCATCAATTTCAACGTCCGGTTCCGTCTGGGCGTGGCGCT
GGACAACCTCGGCCGTTTCGACGAAGCCATCGACAGCTTCAAGATCGCGCTGGGCCTGCGTCCCAA
TGAAGGCAAGGTGCATCGCGCCATCGCCTTCAGCTATGAGCAGATGGGCCGGCACGAGGAAGCCTT
GCCGCATTTCAAGAAGGCCAATGAACTTGACGAAGGGGCCTCGGTCTGAGGCCGTATGTGGGGCGT
TTCCTTCATACGACGAGCTTAGAGAGGCGGATATGGCATTAGGCGACGCGAATGTTGGTTCGGCCC
CTGGGGTCGACTTCAGAA
my_seq2.translate()
PTITRDMVARSVNPHEVRGPXEACHVIK*SEPXD*DFGEHYV*QAVGYS*RGHSLRSLRPFGGEEA
RNEHGRCVPCGLFRQRRHPPGVLPRQGHLPRQGRALFPGRHAAGAGLRRRCLRCGCGSAPGNRLCE
DRRRRSRHRTTRTFLGRCARQREGGDRSRPDLCAGAKVRSGRSAADQGGRGQSHQFQRPVPSGRGA
GQPRPFRRSHRQLQDRAGPASQ*RQGASRHRLQL*ADGPARGSLAAFQEGQ*T*RRGLGLRPYVGR
FLHTTSLERRIWH*ATRMLVRPLGSTSE
```

Ще один спосіб вибору кроку, який Ви могли бачити з рядком python, - це використання кроку -1 для обернення порядку літер рядка. Ви також можете зробити з екземпляром my_seq класу Seq:

```
# Обернення порядку літер послідовності
print(my_seq[::-1])
```

```
Результат роботи коду:
GACTTCAGCTGGGGTCCCCGGCTTGGTTGTAAGCGCAGCGGATTACGGTATAGGCGGAGAGATTCG
AGCAGCATACTTCCTTTGCGGGGTGTATGCCGGAGTCTGGCTCCGGGGAAGCAGTTCAAGTAACCG
GAAGAACTTTACGCCGTTCCGAAGGAGCACGGCCGGGTAGACGAGTATCGACTTCCGCTACCGCGC
TACGTGGAACGGAAGTAACCCTGCGTCCGGGTCGCGCTAGAACTTCGACAGCTACCGAAGCAGCTT
TGCCGGCTCCAACAGGTCGCGGTGCGGGTCTGCCTTGGCCTGCAACTTTAACTACCCTAACCGGAG
CCGGTGGAACTAGTCGTCGCCTTGCCGGTCTAGCATGAAAACGTGGACGTGTATCCAGTCCGGCTC
TTGCCAGCGGTGGAAGTGCAACAGCCCGCGTAGCCGGTTCCTTGCAAGCTCATCAAGCCACGGCGC
TAGCTGCCGCGGCCAGAAGTGTATCCGCTAAGGGTCCACGTCTCGGTGTAGGTGTAGCTTCCGTAG
CCGCAGCATCTGGACGAGGTCGTCGTACTGCCGGACCCTTATCGCGGGCCGGAACCGCACCCTCTA
CGGGAACAGCGCCATTATGTGGACCGCCTACAGCAGCAACTGCCTTTTCCGGCGTGCCTTGCGTAG
CTGGTACAAGTAAGGCTCGAAGAAGCGGTGGCTTTCCGGCATCACTCGCATTTCTCACTGGAGCAG
```

```
TTCTTATAGGCTGCCGAACGATCTGTATCACAAGAGGTTTTAGGATTAGTNCACCGAGACTAGTGA
AATACTGCACCGTCCGAAGNGTGCCGGGCGCGTGAAGCACTCCTAACTGCGAGGCCCGCTGGTATA
GTGCGCACTAGCAGCCGC
```

Наступний приклад показує спосіб завдання регістру літер послідовності:

```
# Завдання регістру літер послідовності
from Bio.Seq import Seq
print(my_seq.upper())
print(my_seq.lower())

Результат роботи коду:
CGCCGACGATCACGCGTGATATGGTCGCCCGGAGCGTCAATCCTCACGAAGTGCGCGGGCCGTGNG
AAGCCTGCCACGTCATAAAGTGATCAGAGCCACNTGATTAGGATTTTGGAGAACACTATGTCTAGC
AAGCCGTCGGATATTCTTGACGAGGTCACTCTTTACGCTCACTACGGCCTTTCGGTGGCGAAGAAG
CTCGGAATGAACATGGTCGATGCGTTCCGTGCGGCCTTTTCCGTCAACGACGACATCCGCCAGGTG
TATTACCGCGACAAGGGCATCTCCCACGCCAAGGCCGGGCGCTATTCCCAGGCCGTCATGCTGCTG
GAGCAGGTCTACGACGCCGATGCCTTCGATGTGGATGTGGCTCTGCACCTGGGAATCGCCTATGTG
AAGACCGGCGCCGTCGATCGCGGCACCGAACTACTCGAACGTTCCTTGGCCGATGCGCCCGACAAC
GTGAAGGTGGCGACCGTTCTCGGCCTGACCTATGTGCAGGTGCAAAAGTACGATCTGGCCGTTCCG
CTGCTGATCAAGGTGGCCGAGGCCAATCCCATCAATTTCAACGTCCGGTTCCGTCTGGGCGTGGCG
CTGGACAACCTCGGCCGTTTCGACGAAGCCATCGACAGCTTCAAGATCGCGCTGGGCCTGCGTCCC
AATGAAGGCAAGGTGCATCGCGCCATCGCCTTCAGCTATGAGCAGATGGGCCGGCACGAGGAAGCC
TTGCCGCATTTCAAGAAGGCCAATGAACTTGACGAAGGGGCCTCGGTCTGAGGCCGTATGTGGGGC
GTTTCCTTCATACGACGAGCTTAGAGAGGCGGATATGGCATTAGGCGACGCGAATGTTGGTTCGGC
CCCTGGGGTCGACTTCAG
Cgccgacgatcacgcgtgatatggtcgcccggagcgtcaatcctcacgaagtgcgcgggccgtgng
aagcctgccacgtcataaagtgatcagagccacntgattaggattttggagaacactatgtctagc
aagccgtcggatattcttgacgaggtcactctttacgctcactacggcctttcggtggcgaagaag
ctcggaatgaacatggtcgatgcgttccgtgcggcctttccgtcaacgacgacatccgccaggtg
tattaccgcgacaagggcatctcccacgccaaggccgggcgctattcccaggccgtcatgctgctg
gagcaggtctacgacgccgatgccttcgatgtggatgtggctctgcacctgggaatcgcctatgtg
aagaccggcgccgtcgatcgcggcaccgaactactcgaacgttccttggccgatgcgcccgacaac
gtgaaggtggcgaccgttctcggcctgacctatgtgcaggtgcaaaagtacgatctggccgttccg
ctgctgatcaaggtggccgaggccaatcccatcaatttcaacgtccggttccgtctgggcgtggcg
ctggacaacctcggccgtttcgacgaagccatcgacagcttcaagatcgcgctgggcctgcgtccc
aatgaaggcaaggtgcatcgcgccatcgccttcagctatgagcagatgggccggcacgaggaagcc
ttgccgcatttcaagaaggccaatgaacttgacgaaggggcctcggtctgaggccgtatgtggggc
gtttccttcatacgacgagcttagagaggcggatatggcattaggcgacgcgaatgttggttcggc
ccctggggtcgacttcag
```

Розглянемо також друк символів послідовності та їх номерів. Функція python enumerate передає індекс символу і сам символ ітерованої послідовності my_seq:

```
# Друк символів послідовності та їх номерів
for index, letter in enumerate(my_seq):
    print(f'index, letter = {index, letter}')
    print(f'len(my_seq) = {len(my_seq)}')

Результат роботи коду:
index, letter = (0, 'C')
index, letter = (1, 'G')
index, letter = (2, 'C')
```

```
index, letter = (3, 'C')
index, letter = (4, 'G')
index, letter = (5, 'A')
index, letter = (6, 'C')
……
index, letter = (870, 'C')
index, letter = (871, 'T')
index, letter = (872, 'T')
index, letter = (873, 'C')
index, letter = (874, 'A')
index, letter = (875, 'G')
len(my_seq) = 876
```

Наступний приклад ілюструє отримання комплементарної послідовності (Рисунок 3.13):

```
# Отримання комплементарної
# послідовності
# екземпляру my_seq класу Seq
from Bio.Seq import Seq
result = my_seq.complement()
print(result)
```

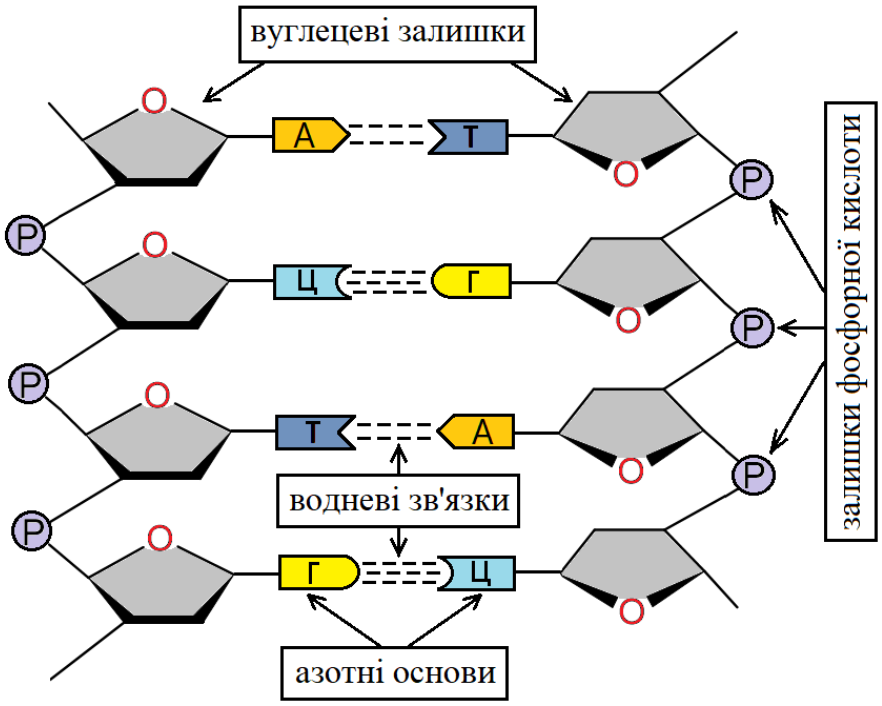


Рисунок 3.13 – Принцип комплементарності.

```
Результат роботи коду:

GCGGCTGCTAGTGCGCACTATACCAGCGGGCCTCGCAGTTAGGAGTGCTTCACGCGCCCGGCACNC
TTCGGACGGTGCAGTATTTCACTAGTCTCGGTGNACTAATCCTAAAACCTCTTGTGATACAGATCG
TTCGGCAGCCTATAAGAACTGCTCCAGTGAGAAATGCGAGTGATGCCGGAAAGCCACCGCTTCTTC
GAGCCTTACTTGTACCAGCTACGCAAGGCACGCCGGAAAAGGCAGTTGCTGCTGTAGGCGGTCCAC
ATAATGGCGCTGTTCCCGTAGAGGGTGCGGTTCCGGCCCGCGATAAGGGTCCGGCAGTACGACGAC
```

```
CTCGTCCAGATGCTGCGGCTACGGAAGCTACACCTACACCGAGACGTGGACCCTTAGCGGATACAC
TTCTGGCCGCGGCAGCTAGCGCCGTGGCTTGATGAGCTTGCAAGGAACCGGCTACGCGGGCTGTTG
CACTTCCACCGCTGGCAAGAGCCGGACTGGATACACGTCCACGTTTTCATGCTAGACCGGCAAGGC
GACGACTAGTTCCACCGGCTCCGGTTAGGGTAGTTAAAGTTGCAGGCCAAGGCAGACCCGCACCGC
GACCTGTTGGAGCCGGCAAAGCTGCTTCGGTAGCTGTCGAAGTTCTAGCGCGACCCGGACGCAGGG
TTACTTCCGTTCCACGTAGCGCGGTAGCGGAAGTCGATACTCGTCTACCCGGCCGTGCTCCTTCGG
AACGGCGTAAAGTTCTTCCGGTTACTTGAACTGCTTCCCCGGAGCCAGACTCCGGCATACACCCCG
CAAAGGAAGTATGCTGCTCGAATCTCTCCGCCTATACCGTAATCCGCTGCGCTTACAACCAAGCCG
GGGACCCCAGCTGAAGTC
```

Нуклеїнові кислоти – це інформаційний банк, у якому знаходяться усі відомості про склад, розвиток і функціонування живих систем (Рисунок 3.14).

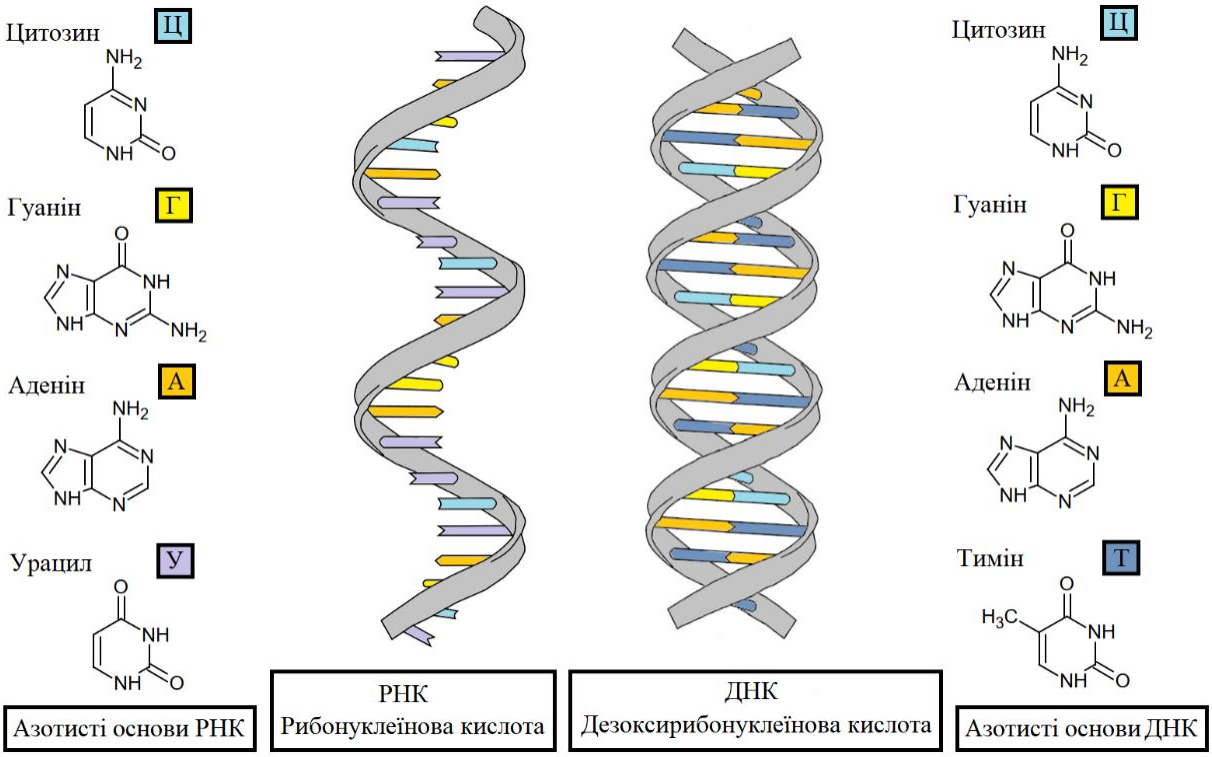


Рисунок 3.14 – Нуклеїнові кислоти.

Зворотно-комплементарна послідовність РНК або ДНК може виникати при утворенні шпильок (Рисунок 3.15) (тобто розворотом вихідної послідовності і взаємодією символів з комплементарними, наприклад, зворотно-комплементарна послідовність: GAAGTCGACCCAACCTCGACTTC).

Покажемо, як методами Biopython можна знайти зворотно-комплементарну послідовність:

```
# Знаходження зворотно-комплементарної послідовності
print(f'my_seq.reverse_complement() = \n
{my_seq.reverse_complement()}')
```

```
Результат роботи коду:
    my_seq.reverse_complement()                                    =
TGAAGTCGACCCCAGGGGCCGAACCAACATTCGCGTCGCCTAATGCCATATCCGCCTCTCTAAGCT
CGTCGTATGAAGGAAACGCCCCACATACGGCCTCAGACCGAGGCCCCTTCGTCAAGTTCATTGGCC
TTCTTGAAATGCGGCAAGGCTTCCTCGTGCCGGCCCATCTGCTCATAGCTGAAGGCGATGGCGCGA
```

```
TGCACCTTGCCTTCATTGGGACGCAGGCCCAGCGCGATCTTGAAGCTGTCGATGGCTTCGTCGAAA
CGGCCGAGGTTGTCCAGCGCCACGCCCAGACGGAACCGGACGTTGAAATTGATGGGATTGGCCTCG
GCCACCTTGATCAGCAGCGGAACGGCCAGATCGTACTTTTGCACCTGCACATAGGTCAGGCCGAGA
ACGGTCGCCACCTTCACGTTGTCGGGCGCATCGGCCAAGGAACGTTCGAGTAGTTCGGTGCCGCGA
TCGACGGCGCCGGTCTTCACATAGGCGATTCCCAGGTGCAGAGCCACATCCACATCGAAGGCATCG
GCGTCGTAGACCTGCTCCAGCAGCATGACGGCCTGGGAATAGCGCCCGGCCTTGGCGTGGGAGATG
CCCTTGTCGCGGTAATACACCTGGCGGATGTCGTCGTTGACGGAAAAGGCCGCACGGAACGCATCG
ACCATGTTCATTCCGAGCTTCTTCGCCACCGAAAGGCCGTAGTGAGCGTAAAGAGTGACCTCGTCA
AGAATATCCGACGGCTTGCTAGACATAGTGTTCTCCAAAATCCTAATCANGTGGCTCTGATCACTT
TATGACGTGGCAGGCTTCNCACGGCCCGCGCACTTCGTGAGGATTGACGCTCCGGGCGACCATATC
ACGCGTGATCGTCGGCG
```

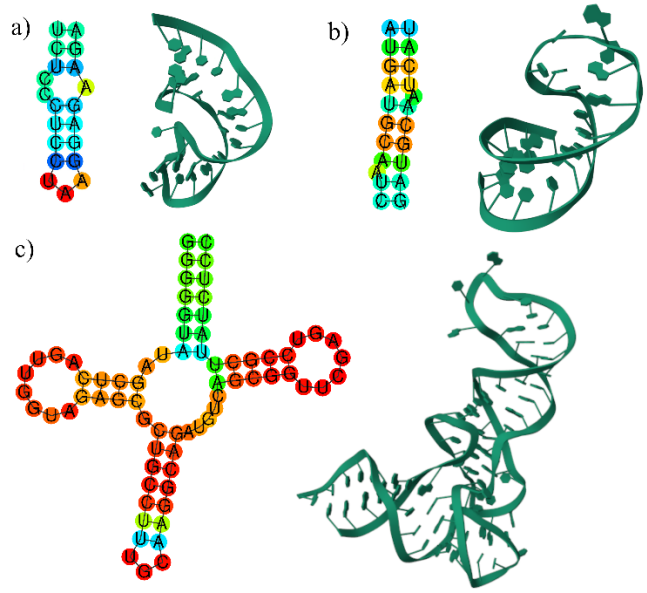


Рисунок 3.15 – Вторинні структури РНК: (a) шпилька з внутрішньої петлею, (b) дуплекс з двома вип'ячуваннями, (c) тРНК. Праворуч показані 3D-моделі, що передбачені за допомогою методів біоінформатики.

Транскрипція – це процес синтезу РНК з використанням ДНК як матриці, що відбувається у всіх живих клітинах, іншими словами, це перенесення генетичної інформації з ДНК на РНК. Однак в Biopython ми зазвичай працюємо безпосередньо з кодуючим ланцюгом і можемо отримати послідовність мРНК, змінивши букву T на U. Наступний приклад показує здійснення транскрипції:

```
# Транскрипція
print(f'my_seq.transcribe() = \n {my_seq.transcribe()}')
```

```
Результат роботи коду:
my_seq.transcribe() =
GCCGACGAUCACGCGUGAUAUGGUCGCCCGGAGCGUCAAUCCUCACGAAGUGCGCGGGCCGUGNGA
AGCCUGCCACGUCAUAAAGUGAUCAGAGCCACNUGAUUAGGAUUUUGGAGAACACUAUGUCUAGCA
AGCCGUCGGAUAUUCUUGACGAGGUCACUCUUUACGCUCACUACGGCCUUUCGGUGGCGAAGAAGC
UCGGAAUGAACAUGGUCGAUGCGUUCCGUGCGGCCUUUUCCGUCAACGACGACAUCCGCCAGGUGU
AUUACCGCGACAAGGGCAUCUCCCACGCCAAGGCCGGGCGCUAUUCCCAGGCCGUCAUGCUGCUGG
AGCAGGUCUACGACGCCGAUGCCUUCGAUGUGGAUGUGGCUCUGCACCUGGGAAUCGCCUAUGUGA
AGACCGGCGCCGUCGAUCGCGGCACCGAACUACUCGAACGUUCCUUGGCCGAUGCGCCCGACAACG
UGAAGGUGGCGACCGUUCUCGGCCUGACCUAUGUGCAGGUGCAAAAGUACGAUCUGGCCGUUCCGC
UGCUGAUCAAGGUGGCCGAGGCCAAUCCCAUCAAUUUCAACGUCCGGUUCCGUCUGGGCGUGGCGC
```

```
UGGACAACCUCGGCCGUUUCGACGAAGCCAUCGACAGCUUCAAGAUCGCGCUGGGCCUGCGUCCCA
AUGAAGGCAAGGUGCAUCGCGCCAUCGCCUUCAGCUAUGAGCAGAUGGGCCGGCACGAGGAAGCCU
UGCCGCAUUUCAAGAAGGCCAAUGAACUUGACGAAGGGGCCUCGGUCUGAGGCCGUAUGUGGGGCG
UUUCCUUCAUACGACGAGCUUAGAGAGGCGGAUAUGGCAUUAGGCGACGCGAAUGUUGGUUCGGCC
CCUGGGGUCGACUUCAG
```

Здійснимо також транскрипцію зворотно-комплементарної послідовності:

```
# Транскрипція зворотно-комплементарної послідовності
print(f'my_seq.reverse_complement().transcribe() = '
      f'\n {my_seq.reverse_complement().transcribe()}')
```

```
Результат роботи коду:
my_seq.reverse_complement().transcribe() =
CUGAAGUCGACCCCAGGGGCCGAACCAACAUUCGCGUCGCCUAAUGCCAUAUCCGCCUCUCUAAGC
UCGUCGUAUGAAGGAAACGCCCCACAUACGGCCUCAGACCGAGGCCCCUUCGUCAAGUUCAUUGGC
CUUCUUGAAAUGCGGCAAGGCUUCCUCGUGCCGGCCCAUCUGCUCAUAGCUGAAGGCGAUGGCGCG
AUGCACCUUGCCUUCAUUGGGACGCAGGCCCAGCGCGAUCUUGAAGCUGUCGAUGGCUUCGUCGAA
ACGGCCGAGGUUGUCCAGCGCCACGCCCAGACGGAACCGGACGUUGAAAUUGAUGGGAUUGGCCUC
GGCCACCUUGAUCAGCAGCGGAACGGCCAGAUCGUACUUUUGCACCUGCACAUAGGUCAGGCCGAG
AACGGUCGCCACCUUCACGUUGUCGGGCGCAUCGGCCAAGGAACGUUCGAGUAGUUCGGUGCCGCG
AUCGACGGCGCCGGUCUUCACAUAGGCGAUUCCCAGGUGCAGAGCCACAUCCACAUCGAAGGCAUC
GGCGUCGUAGACCUGCUCCAGCAGCAUGACGGCCUGGGAAUAGCGCCCGGCCUUGGCGUGGGAGAU
GCCCUUGUCGCGGUAAUACACCUGGCGGAUGUCGUCGUUGACGGAAAAGGCCGCACGGAACGCAUC
GACCAUGUUCAUUCCGAGCUUCUUCGCCACCGAAAGGCCGUAGUGAGCGUAAAGAGUGACCUCGUC
AAGAAUAUCCGACGGCUUGCUAGACAUAGUGUUCUCCAAAAUCCUAAUCANGUGGCUCUGAUCACU
UUAUGACGUGGCAGGCUUCNCACGGCCCGCGCACUUCGUGAGGAUUGACGCUCCGGGCGACCAUAU
CACGCGUGAUCGUCGGCG
```

Зворотна транскрипція – це процес синтезу ДНК з використанням РНК як матриці, називається так тому, що в більшості живих організмів транскрипція відбувається в іншому напрямку, а саме — з молекули ДНК синтезується РНК-транскрипт. Зворотна транскрипція вперше встановлена для одного з класів РНК-вмісних вірусів тварин (ретровірусів) і забезпечується вона спеціальним ферментом — зворотною транскриптазою. Зворотна транскриптаза (РНК-залежна ДНК-полімераза, або ревертаза, КФ 2.7.7.49) відкрита Говардом М. Тьоміном та Девідом Балтимором у 1970 р., за це відкриття вчені отримали Нобелівську премію з біології й медицини (1976). Після того як ретровірус потрапляє в клітину хазяїна, відбувається його розмноження. За допомогою ревертази спочатку до матриці РНК вірусу комплементарно приєднуються дезоксирибонуклеозидтрифосфати і синтезується один ланцюг ДНК. При цьому утворюється гібридна об’єднана молекула РНК-ДНК. Потім фермент РНКаза Н видаляє ланцюг рибонуклеотидів із гібридної молекули, а на ланцюзі ДНК комплементарно за наявності ферменту ДНК-полімерази здійснюється синтез другого ланцюга ДНК. Наступний код здійснює зворотну транскрипцію:

```
# Зворотна транскрипція
print(f'my_seq.transcribe().back_transcribe() = \n'
      f' {my_seq.transcribe().back_transcribe()}')
```

```
Результат роботи коду:
my_seq.transcribe().back_transcribe()                                    =
TGAAGTCGACCCCAGGGGCCGAACCAACATTCGCGTCGCCTAATGCCATATCCGCCTCTCTAAGCT
CGTCGTATGAAGGAAACGCCCCACATACGGCCTCAGACCGAGGCCCCTTCGTCAAGTTCATTGGCC
TTCTTGAAATGCGGCAAGGCTTCCTCGTGCCGGCCCATCTGCTCATAGCTGAAGGCGATGGCGCGA
```

```
TGCACCTTGCCTTCATTGGGACGCAGGCCCAGCGCGATCTTGAAGCTGTCGATGGCTTCGTCGAAA
CGGCCGAGGTTGTCCAGCGCCACGCCCAGACGGAACCGGACGTTGAAATTGATGGGATTGGCCTCG
GCCACCTTGATCAGCAGCGGAACGGCCAGATCGTACTTTTGCACCTGCACATAGGTCAGGCCGAGA
ACGGTCGCCACCTTCACGTTGTCGGGCGCATCGGCCAAGGAACGTTCGAGTAGTTCGGTGCCGCGA
TCGACGGCGCCGGTCTTCACATAGGCGATTCCCAGGTGCAGAGCCACATCCACATCGAAGGCATCG
GCGTCGTAGACCTGCTCCAGCAGCATGACGGCCTGGGAATAGCGCCCGGCCTTGGCGTGGGAGATG
CCCTTGTCGCGGTAATACACCTGGCGGATGTCGTCGTTGACGGAAAAGGCCGCACGGAACGCATCG
ACCATGTTCATTCCGAGCTTCTTCGCCACCGAAAGGCCGTAGTGAGCGTAAAGAGTGACCTCGTCA
AGAATATCCGACGGCTTGCTAGACATAGTGTTCTCCAAAATCCTAATCANGTGGCTCTGATCACTT
TATGACGTGGCAGGCTTCNCACGGCCCGCGCACTTCGTGAGGATTGACGCTCCGGGCGACCATATC
ACGCGTGATCGTCGGCG
```

Трансляція – це синтез білків з амінокислот, що каталізується рибосомою на матриці матричної РНК (мРНК) (Рисунок 3.16). Трансляція є однією зі стадій біосинтезу білків, а він, у свою чергу – частина процесу експресії генів.

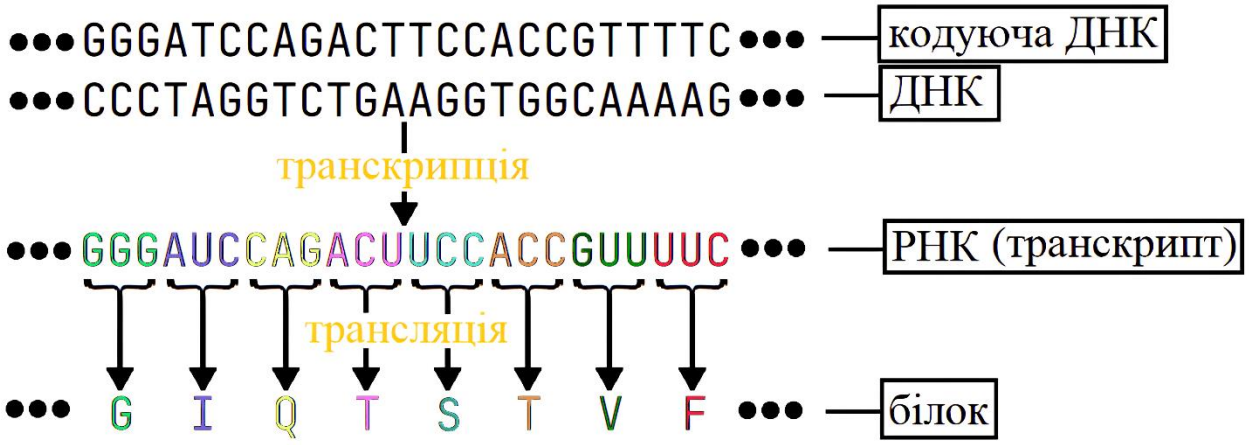


Рисунок 3.16 – Схематичне зображення експресії генів.

Наступний код здійснює трансляцію:

```
# Трансляція
print(f'my_seq.translate() = \n {my_seq.translate()}')
print(str(len(my_seq)))
print(str(len(my_seq.translate())))

Результат роботи коду:
my_seq.translate() =
RRSRVIWSPGASILTKCAGRXKPATS*SDQSHXIRILENTMSSKPSDILDEVTLYAHYGLSVAKKL
GMNMVDAFRAAFSVNDDIRQVYYRDKGISHAKAGRYSQAVMLLEQVYDADAFDVDVALHLGIAYVK
TGAVDRGTELLERSLADAPDNVKVATVLGLTYVQVQKYDLAVPLLIKVAEANPINFNVRFRLGVAL
DNLGRFDEAIDSFKIALGLRPNEGKVHRAIAFSYEQMGRHEEALPHFKKANELDEGASV*GRMWGV
SFIRRA*RGGYGIRRRECWFGPWGRLQ
```

Відомо, що у послідовностях мРНК зустрічаються стоп-кодони або кодони термінації – трійки нуклеотидних залишків в мРНК, що кодують припинення (термінацію) синтезу поліпептидного ланцюга, тобто припинення трансляції. Стандартні стоп-кодони – це UAA, UAG, UGA. Загалом у послідовностях, крім символу кінцевого стоп-кодону, може зустрічатися також внутрішня термінація. В такому випадку важливо знати деякі

необов'язкові аргументи функції translate(), включаючи різні таблиці трансляції (генетичні коди).

Наступний приклад здійснює трансляцію РНК з урахуванням стандартного генетичного коду:

```
# Трансляція РНК з урахуванням стандартного генетичного коду
from Bio.Seq import Seq
coding_rna = Seq ("AUGGCCAUUGUAAUGGCCGCUGAAAGGGUGCCCGAUAGA")
print(f'coding_rna = {coding_rna}')
coding_rna.translate()
print(f'coding_rna_translate = {coding_rna.translate()}')

Результат роботи коду: coding_rna =
AUGGCCAUUGUAAUGGCCGCUGAAAGGGUGCCCGAUAGA
coding_rna_translate = MAIVMAAERVPD
```

Наступний приклад здійснює трансляцію ДНК з урахуванням стандартного генетичного коду:

```
# Трансляція ДНК з урахуванням стандартного генетичного коду
from Bio.Seq import Seq
coding_dna = Seq ("ATGGCCATTGTAATGGCCGCTGAAAGGGTGCCCGATAG")
print(f'coding_dna = {coding_dna}')
coding_dna.translate()
print(f'coding_dna_translate = {coding_dna.translate()}')
print(len(coding_dna))
print(len(coding_dna.translate()))

Результат роботи коду:
coding_dna = ATGGCCATTGTAATGGCCGCTGAAAGGGTGCCCGATAG
coding_dna_translate = MAIVMAAERVPD
39
13
```

Для більшості еукаріотів зазвичай використовується стандартна таблиця трансляції, у якій кожній амінокислоті відповідає одна або кілька послідовностей ДНК (див. NCBI Taxonomy).

Не всі організми використовують однаковий генетичний код. Навіть працюючи із генетичними послідовностями звичайних еукаріотичних організмів, наприклад дріжджів, часто бажано використовувати альтернативні таблиці генетичного коду, наприклад, для трансляції мітохондріальних геномів.

На сьогодні група таксономії NCBI визначає таблиці кодів для послідовностей, що містить GenBank.

В пакеті Biopython прийнято таку нумерацію таблиць генетичного коду:

1: Стандартний код
2: Мітохондріальний код хребетних
3: Мітохондріальний код дріжджів
4: Мітохондріальний код слизистих грибків, протозоїв і кишковопорожнинних та мікоплазми
5: Мітохондріальний код безхребетних
6: Ядерний код джгутикових, Dasycladacea і Hexamita
9: Мітохондріальний код голошкірих та плоских червів
10: Ядерний код Euplotida
11: Бактеріальний код і пластидний код рослин

12: Альтернативний ядерний код дріжджів
13: Мітохондріальний код асцидій
14: Альтернативний мітохондріальний код плоских червів
15: Ядерний код Blepharisma
16: Мітохондріальний код Chlorophycea
21: Мітохондріальний код Trematoda
22: Мітохондріальний код Scenedesmus obliquus
23: Мітохондріальний код Thraustochytrium

Таблиці генетичного коду, доступні в Biopython, базуються на таблицях NCBI. За замовчуванням для трансляції буде використовуватися стандарт-ний генетичний код (таблиця NCBI ідентифікатор 1). Припустимо, ми маємо справу з мітохондріальною послідовністю. Нам потрібно в функції translate() використовувати замість ідентифікатора 1 відповідний генетичний код. Ви також можете вказати таблицю (наприклад, table = 'Vertebrate Mitochondrial‘ (Мітохондріальний код хребетних), використовуючи номер таблиці NCBI, який є коротшим і часто включеним до анотації функцій файлів GenBank: coding_dna.translate (table = 2). В наступному прикладі трансляція здійснюється з урахуванням таблиць генетичного коду мітохондрій хребетних:

```
# Трансляції з урахуванням таблиць генетичного коду мітохондрій
хребетних
print(f"coding_dna_translate table='Vertebrate Mitochondrial = "
      f"{coding_dna.translate(table='Vertebrate Mitochondrial')}")
Результат роботи коду:
coding_dna_translate table='Vertebrate Mitochondrial =
MAIVMAAE*VPD
MAIVMAAERVPD
```

В пакеті Biopython можна надрукувати таблицю трансляції. Стандартна таблиця генетичного коду виводиться на друк в наступному прикладі (Рисунок 3.17):

```
# Стандартна таблиця генетичного коду
# модуль CodonTable пакету Bio.Data
from Bio.Data import CodonTable
standard_table =   CodonTable.unambiguous_dna_by_id[1]
print(f'standard_table = {standard_table}')
```

- В попередньому прикладі:
- 1 - це стандартний код;
- unambiguous – однозначний код (Таблиця 3.1);
- ambiguous – неоднозначний код, коли не відомо, який нуклеотид в певній позиції (Таблиця 3.1);
- unambiguous_dna_by_id – словник (ключ-номер таблиці, значення – сама таблиця)

Таблиця 3.1. Таблиця unambiguous (однозначного коду) та ambiguous (неоднозначного коду).

IUPAC Code	Meaning	Complement
A	A	T
C	C	G
G	G	C
T/U	T	A
M	A or C	K
R	A or G	Y
W	A or T	W
S	C or G	S
Y	C or T	R
K	G or T	M
V	A or C or G	B
H	A or C or T	D
D	A or G or T	H
B	C or G or T	V
N	G or A or T or C	N

```
  |  T       |  C       |  A       |  G       |
--+----------+----------+----------+----------+--
T | TTT F    | TCT S    | TAT Y    | TGT C    | T
T | TTC F    | TCC S    | TAC Y    | TGC C    | C
T | TTA L    | TCA S    | TAA Stop| TGA Stop| A
T | TTG L(s)| TCG S    | TAG Stop| TGG W    | G
--+----------+----------+----------+----------+--
C | CTT L    | CCT P    | CAT H    | CGT R    | T
C | CTC L    | CCC P    | CAC H    | CGC R    | C
C | CTA L    | CCA P    | CAA Q    | CGA R    | A
C | CTG L(s)| CCG P    | CAG Q    | CGG R    | G
--+----------+----------+----------+----------+--
A | ATT I    | ACT T    | AAT N    | AGT S    | T
A | ATC I    | ACC T    | AAC N    | AGC S    | C
A | ATA I    | ACA T    | AAA K    | AGA R    | A
A | ATG M(s)| ACG T    | AAG K    | AGG R    | G
--+----------+----------+----------+----------+--
G | GTT V    | GCT A    | GAT D    | GGT G    | T
G | GTC V    | GCC A    | GAC D    | GGC G    | C
G | GTA V    | GCA A    | GAA E    | GGA G    | A
G | GTG V    | GCG A    | GAG E    | GGG G    | G
--+----------+----------+----------+----------+--
```

Рисунок 3.17 – Результат роботи коду із попереднього прикладу.

Таблиця генетичного коду у випадку мітохондріального коду хребетних виводиться на друк в наступному прикладі (Рисунок 3.18):

```
# Таблиця генетичного коду
# Мітохондріальний код хребетних
from Bio.Data import CodonTable
mito_table = CodonTable.unambiguous_dna_by_id[2]
print(f'mito_table = {mito_table}')
```

В попередньому прикладі індекс 2 – це означає мітохондріальний код хребетних. Роздрукуємо також таблицю генетичного коду для випадку бактеріального коду і пластидного коду рослин (Рисунок 3.19):

```
# Таблиця генетичного коду. Бактеріальний код і
# пластидний код рослин.
```

```
from Bio.Data import CodonTable
table11 = CodonTable.unambiguous_dna_by_id[11]
print(f'Бактеріальний код і '
      f'пластидний код рослин table = {table11}')
```

```
  |  T      |  C      |  A      |  G      |
--+---------+---------+---------+---------+--
T | TTT F   | TCT S   | TAT Y   | TGT C   | T
T | TTC F   | TCC S   | TAC Y   | TGC C   | C
T | TTA L   | TCA S   | TAA Stop| TGA W   | A
T | TTG L   | TCG S   | TAG Stop| TGG W   | G
--+---------+---------+---------+---------+--
C | CTT L   | CCT P   | CAT H   | CGT R   | T
C | CTC L   | CCC P   | CAC H   | CGC R   | C
C | CTA L   | CCA P   | CAA Q   | CGA R   | A
C | CTG L   | CCG P   | CAG Q   | CGG R   | G
--+---------+---------+---------+---------+--
A | ATT I(s)| ACT T   | AAT N   | AGT S   | T
A | ATC I(s)| ACC T   | AAC N   | AGC S   | C
A | ATA M(s)| ACA T   | AAA K   | AGA Stop| A
A | ATG M(s)| ACG T   | AAG K   | AGG Stop| G
--+---------+---------+---------+---------+--
G | GTT V   | GCT A   | GAT D   | GGT G   | T
G | GTC V   | GCC A   | GAC D   | GGC G   | C
G | GTA V   | GCA A   | GAA E   | GGA G   | A
G | GTG V(s)| GCG A   | GAG E   | GGG G   | G
--+---------+---------+---------+---------+--
```

Рисунок 3.18 – Результат роботи коду із попереднього прикладу.

```
Бактеріальний код і пластидний код рослин table
= Table 11 Bacterial, Archaeal, Plant Plastid
  |  T      |  C      |  A      |  G      |
--+---------+---------+---------+---------+--
T | TTT F   | TCT S   | TAT Y   | TGT C   | T
T | TTC F   | TCC S   | TAC Y   | TGC C   | C
T | TTA L   | TCA S   | TAA Stop| TGA Stop| A
T | TTG L(s)| TCG S   | TAG Stop| TGG W   | G
--+---------+---------+---------+---------+--
C | CTT L   | CCT P   | CAT H   | CGT R   | T
C | CTC L   | CCC P   | CAC H   | CGC R   | C
C | CTA L   | CCA P   | CAA Q   | CGA R   | A
C | CTG L(s)| CCG P   | CAG Q   | CGG R   | G
--+---------+---------+---------+---------+--
A | ATT I(s)| ACT T   | AAT N   | AGT S   | T
A | ATC I(s)| ACC T   | AAC N   | AGC S   | C
A | ATA I(s)| ACA T   | AAA K   | AGA R   | A
A | ATG M(s)| ACG T   | AAG K   | AGG R   | G
--+---------+---------+---------+---------+--
G | GTT V   | GCT A   | GAT D   | GGT G   | T
G | GTC V   | GCC A   | GAC D   | GGC G   | C
G | GTA V   | GCA A   | GAA E   | GGA G   | A
G | GTG V(s)| GCG A   | GAG E   | GGG G   | G
--+---------+---------+---------+---------+--
```

Рисунок 3.19 – Результат роботи коду із попереднього прикладу.

Властивості таблиці трансляції можуть бути корисними, наприклад, якщо Ви намагаєтесь самостійно знайти гени. Функція forward_table модулю CodonTable, пакету Bio.Data повертає результати трансляції кодонів у різних таблицях генетичного коду:

```
# Стоп-, старт-та кодони амінокислот в таблиці
# генетичного мітохондріальний коду хребетних from Bio.Data import
CodonTable
from Bio.Data import CodonTable
mito_table = CodonTable.unambiguous_dna_by_id[2]
print(f'mito_table.stop_codons = {mito_table.stop_codons}')
print(f'mito_table.start_codons = {mito_table.start_codons}')
print(f'mito_table.forward_table ["ACG"] =
{mito_table.forward_table["ACG"]}')    # ACG = T (треонін)

Результат роботи коду:
mito_table.stop_codons = ['TAA', 'TAG', 'AGA', 'AGG']
mito_table.start_codons = ['ATT', 'ATC', 'ATA', 'ATG', 'GTG']
mito_table.forward_table ["ACG"] = T
```

3.5. Попарне вирівнювання послідовностей в Biopython

Наступний код показує, як можна здійснити попарне вирівнювання послідовностей в Biopython:

```
# імпортуємо пакет Align із пакету Bio
from Bio import Align
# створюємо екземпляр aligner класу PairwiseAligner () із
# атрибутами (тобто параметрами вирівнювання, які визначають
систему премій і штрафів) за замовчуванням
aligner = Align.PairwiseAligner()
# задаємо послідовність-мішень - target
target = "GAACT"
# задаємо послідовність-запит - query
query = "GAT"
# задаємо режим вирівнювання
aligner.mode = 'local'
# здійснюємо вирівнювання за допомогою методу align()
# alignments = aligner.align (target, query)
# друкуємо всі отримані варіанти вирівнювань з max вагою
# for alignment in alignments:
# print(alignment)
# підраховуємо та друкуємо вагу вирівнювання за
# допомогою метода score()
score = aligner.score(target, query)
print(f'score= {score}')

Результат роботи коду:
score = 3.0
GAACT
||--|
GA--T
```

```
GAACT
|-|-|
G-A-T
```

В наступному прикладі здійснюється друк атрибутів екземпляру aligner класу PairwiseAligner() (тобто параметрів вирівнювання, які визначають систему премій і штрафів) за замовчуванням:

```
print(aligner)
```

```
Результат роботи коду:
Pairwise sequence aligner with parameters
  match_score: 1.000000 (бали за співпадіння)
  mismatch_score: 0.000000 (штраф за не співпадіння)
  target_internal_open_gap_score:   0.000000   (штраф   за   перший
внутрішній пробіл у послідовності target – тобто у першій)
  target_internal_extend_gap_score:  0.000000  (штраф  за  кожний
наступний внутрішній пробіл у послідовності target)
  target_left_open_gap_score:  0.000000  (штраф  за  перший  лівий
пробіл у послідовності target)
  target_left_extend_gap_score: 0.000000 (штраф за перший кожний
наступний лівий пробіл у послідовності target)
  target_right_open_gap_score:  0.000000  (штраф  за  перший  правий
пробіл у послідовності target)
  target_right_extend_gap_score: 0.000000 (штраф за перший кожний
наступний правий пробіл у послідовності target)
  query_internal_open_gap_score:   0.000000   (штраф   за   перший
внутрішній пробіл у послідовності query – тобто у другій)
  query_internal_extend_gap_score:   0.000000   (штраф   за   кожний
наступний внутрішній пробіл у послідовності query)
  query_left_open_gap_score:  0.000000  (штраф  за  перший  лівий
пробіл у послідовності query)
  query_left_extend_gap_score: 0.000000 (штраф за кожний наступний
лівий пробіл у послідовності query)
  query_right_open_gap_score:  0.000000  (штраф  за  перший  правий
пробіл у послідовності query)
  query_right_extend_gap_score:   0.000000   (штраф   за   кожний
наступний правий пробіл у послідовності query)
  mode: global (режим вирівнювання - глобальний)
```

В Biopython можливе здійснення попарного вирівнювання генетичних послідовностей після зміни параметрів вирівнювання. Наступний приклад показує зміну параметрів вирівнювання:

```
# Здійснення попарного вирівнювання
# генетичних послідовностей
# після зміни параметрів вирівнювання
aligner.query_internal_open_gap_score = -2
aligner.target_internal_open_gap_score = -2
aligner.mismatch_score = -1
print(aligner)
```

```
Результат роботи коду:
```

```
GAACT
||
GAT
Pairwise sequence aligner with parameters
  match_score: 1.000000
  mismatch_score: -1.000000
  target_internal_open_gap_score: -2.000000
  target_internal_extend_gap_score: 0.000000
  target_left_open_gap_score: 0.000000
  target_left_extend_gap_score: 0.000000
  target_right_open_gap_score: 0.000000
  target_right_extend_gap_score: 0.000000
  query_internal_open_gap_score: -2.000000
  query_internal_extend_gap_score: 0.000000
  query_left_open_gap_score: 0.000000
  query_left_extend_gap_score: 0.000000
  query_right_open_gap_score: 0.000000
  query_right_extend_gap_score: 0.000000
  mode: local
```

Пакет Bio.Align, модуль AlignIO, клас MultipleSeqAlignment дозволяє записати результати виконання вирівнювання послідовностей в програмі BLAST або COBALT у файл, наприклад, "my_example.phy" , наприклад:

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio.Align import MultipleSeqAlignment
align1 = MultipleSeqAlignment([SeqRecord(Seq("ACTGCTAGCTAG"),
id="Alpha"),
                               SeqRecord(Seq("ACT-CTAGCTAG"),
id="Beta"),
                               SeqRecord(Seq("ACTGCTAGDTAG"),
id="Gamma"),])
align2 = MultipleSeqAlignment([SeqRecord(Seq("GTCAGC-AG"),
id="Delta"),
                               SeqRecord(Seq("GACAGCTAG"),
id="Epsilon"),
                               SeqRecord(Seq("GTCAGCTAG"),
id="Zeta"),])
align3 = MultipleSeqAlignment([SeqRecord(Seq("ACTAGTACAGCTG"),
id="Eta"),
                               SeqRecord(Seq("ACTAGTACAGCT-"),
id="Theta"),
                               SeqRecord(Seq("-CTACTACAGGTG"),
id="Iota"),])
my_alignments = [align1, align2, align3]
from Bio import AlignIO
AlignIO.write(my_alignments, "my_example.phy", "phylip")
```

Формат phylip використовується для запису вирівняних послідовностей:

```
3 12
Alpha           ACTGCTAGCTAG
Beta            ACT-CTAGCTAG
```

```
Gamma       ACTGCTAGDTAG
 3  9
Delta       GTCAGC-AG
Epsilon     GACAGCTAG
Zeta        GTCAGCTAG
 3  13
Eta         ACTAGTACAGCTG
Theta       ACTAGTACAGCT-
Iota        -CTACTACAGGTG
```

При цьому файл містить:

- кількість послідовностей = 3
- довжину вирівнювання, в даних прикладах 12, 9, 13
- назви послідовностей
- результати множинного вирівнювання

В наступному прикладі ми здійснюємо вирівнювання послідовності AY508230.1 з геномом людини. Спочатку знаходимо послідовність AY508230.1 в NCBI. Потім вибираємо Run BLAST (Рисунок 3.20).

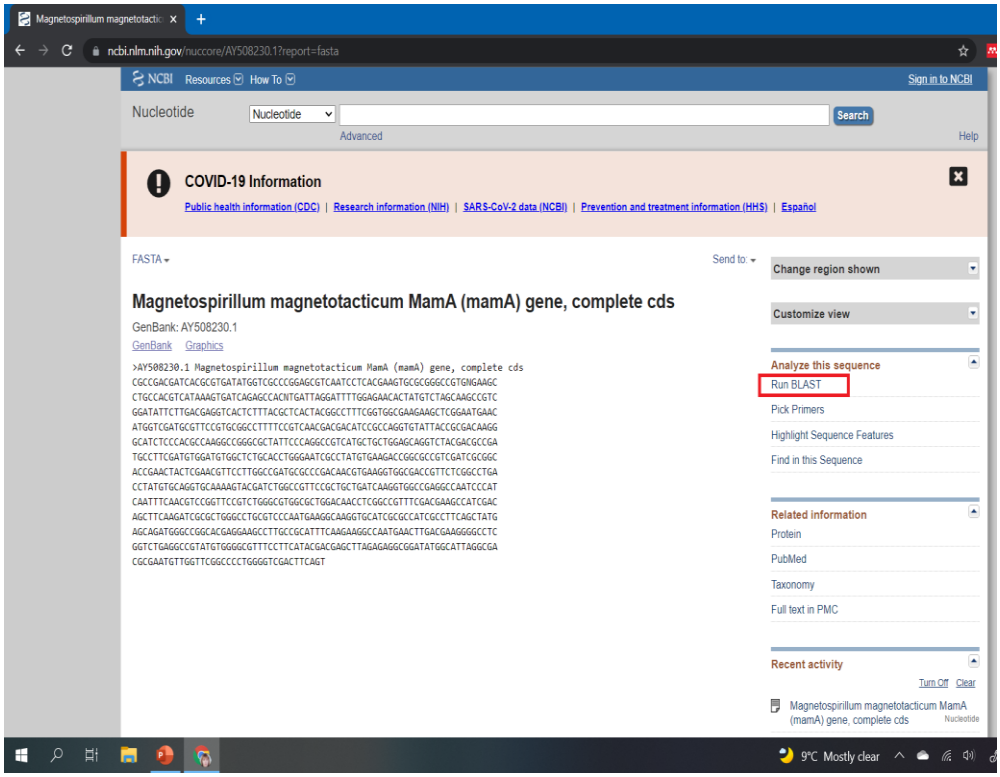


Рисунок 3.20 – Схематичне зображення вибору Run BLAST.

Вибираємо організм human. В параметрах вибираємо (Рисунок 3.21)

- word size = 16,
- Expect
- Threshold = 1000.

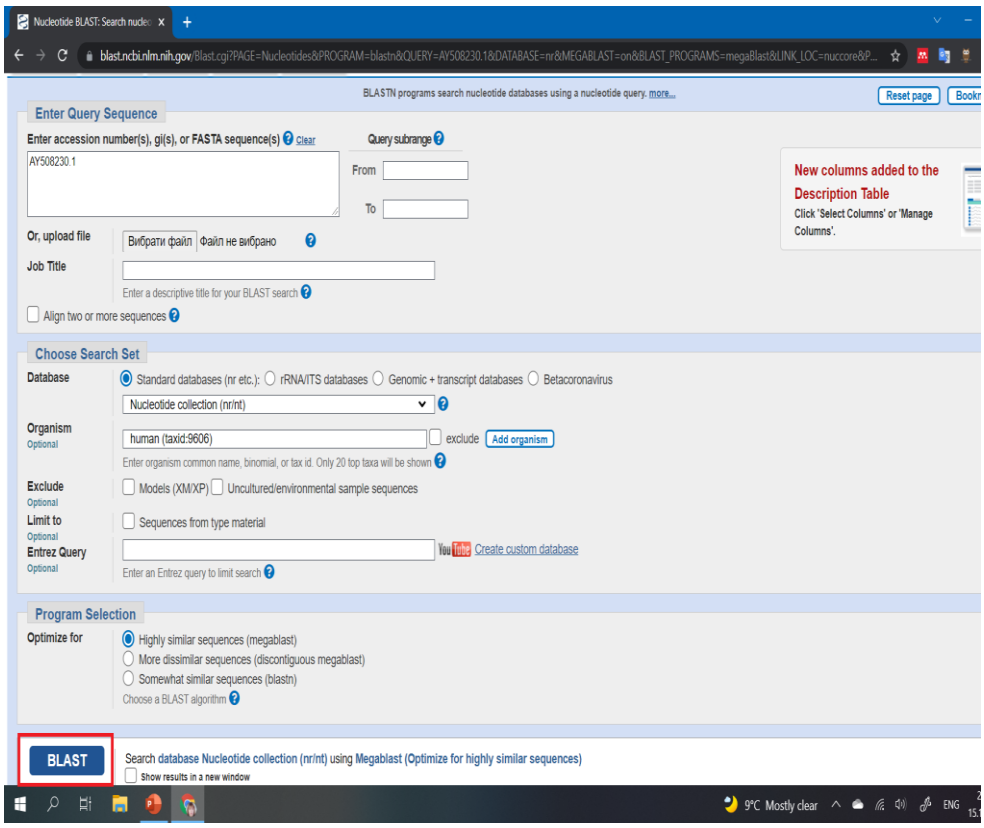


Рисунок 3.21 – Вибір параметрів вирівнювання та організму.

Для завантаження результатів вирівнювання послідовності AY508230.1 з геномом людини вибираємо Download All (Рисунок 3.22).

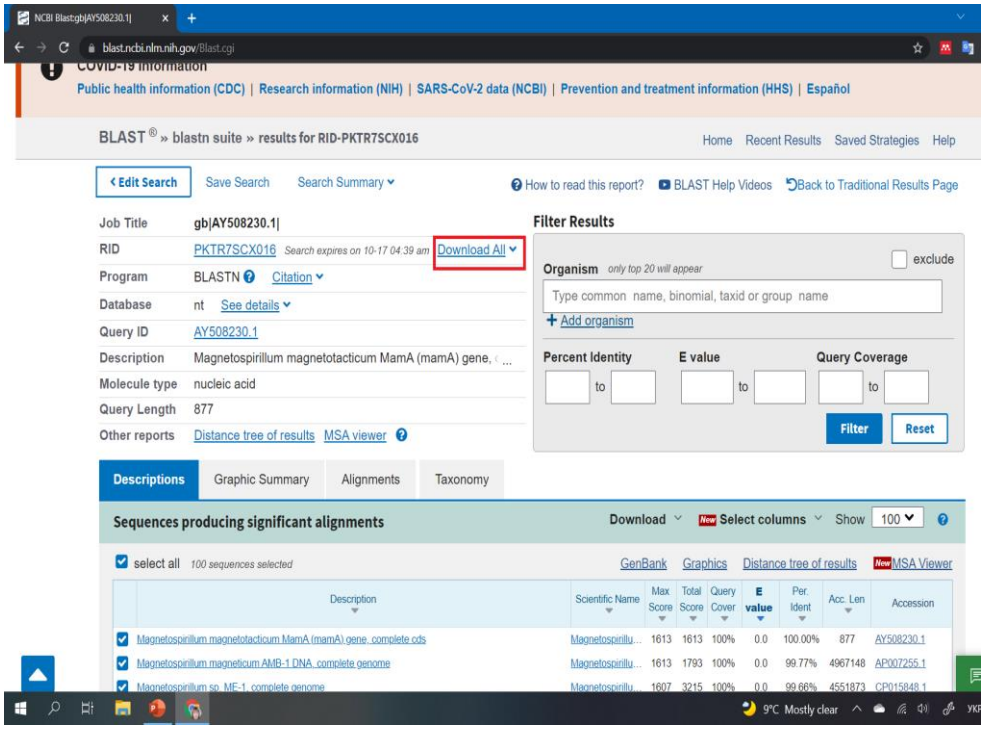


Рисунок 3.22 – Завантаження результатів вирівнювання послідовності AY508230.1 з геномом людини.

Вибираємо тип файлу з результатами вирівнювання XML (Рисунок 3.23).

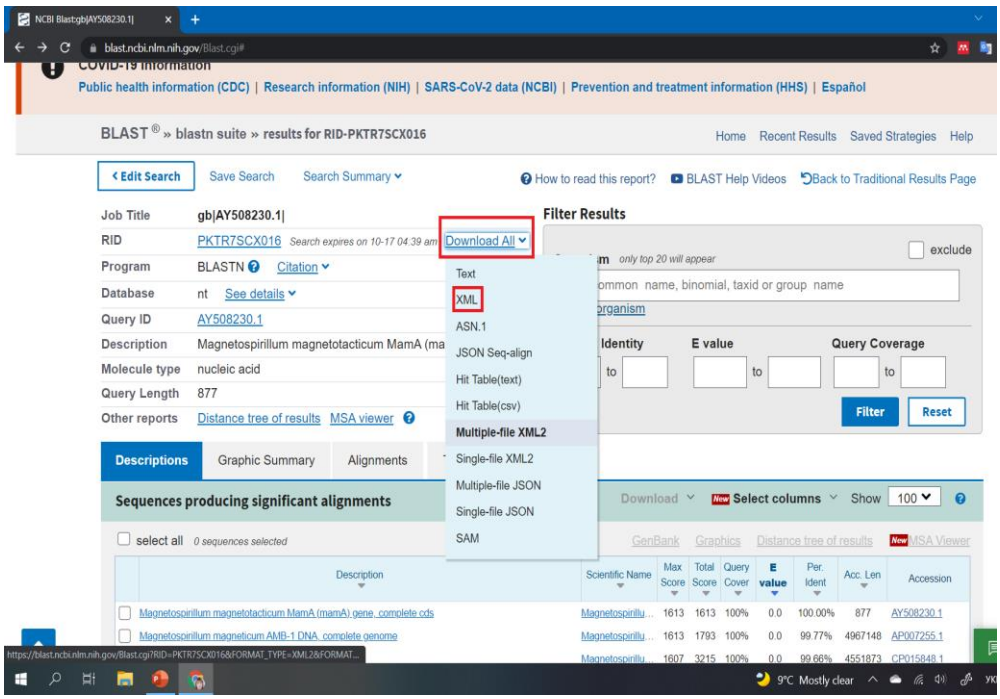


Рисунок 3.23 – Вибір типу файлу з результатами вирівнювання XML.

Завантажуємо файл з результатами вирівнювання і зберігаємо у папці, яку створюємо всередині папки, в якій зберігається наш код (Рисунок 3.24).

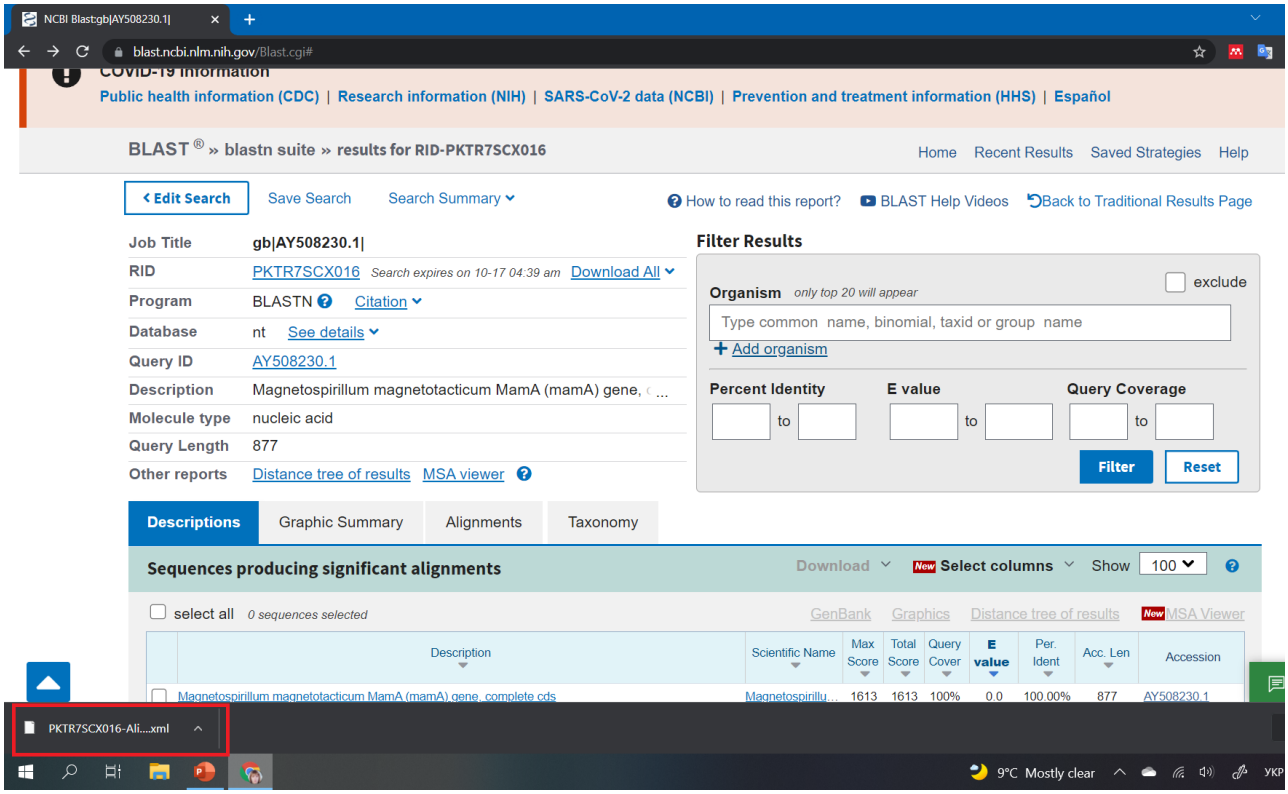


Рисунок 3.24 – Завантаження та збереження файлу з результатами вирівнювання.

3.6. Робота з файлами програми Blast

Як показано в вище, щоб згенерувати файл з результатами вирівнювання в Blast, необхідно з застосуванням програми Blast на сайті NCBI вирівняти послідовності та натиснути кнопку Download All, а потім кнопку xml.

Функція read в пакеті Bio.SearchIO зчитує файл, який створено автоматично в програмі Blast. Зчитування інформації про результати вирівнювання зі збереженого файлу PKTR75CX016-Alignment.xml і її друк:

```
# Робота з файлами програми Blast
# PKTR75CX016-Alignment.xml - файл, який створено автоматично в
програмі Blast
from Bio import SearchIO
blast_qresult = SearchIO.read("BioSeqExamples\PKTR75CX016-
Alignment.xml", "blast-xml")
print(f'blast_qresult = \n {blast_qresult}')

Результат роботи коду:
blast_qresult =
 Program: blastp (2.12.0+)
  Query: WP_106001333.1 (297)
         magnetosome biogenesis CDF transporter MamB
[Magnetospirillum gryphi...
 Target: nr
   Hits: ----  -----  ----------------------------------------
--------------
            #  # HSP  ID + description
          ----  -----  ----------------------------------------
--------------
            0      1  gb|AAP83846.1|  chromosome 4 open reading
frame 1 [Homo...
            1      1  gb|AAH22981.1|  SLC30A9 protein, partial
[Homo sapiens]
            2      1  dbj|BAG61754.1|  unnamed protein product
[Homo sapiens]
            3      1  dbj|BAG35692.1|  unnamed protein product
[Homo sapiens]
            4      1  gb|AAB87763.2|  embryonic lung protein [Homo
sapiens]
            5      1  ref|XP_016863143.1|  zinc transporter 9
isoform X1 [Hom...
            6      1  ref|NP_006336.3|  zinc transporter 9 [Homo
sapiens]
            7      1  gb|AAY40966.1|  unknown, partial [Homo
sapiens]
            8      1  gb|EAW92998.1|  solute carrier family 30
(zinc transpor...
            9      1  gb|AAB82561.1|  zinc transporter 4 [Homo
sapiens]
           10      1  ref|NP_037441.2|  zinc transporter 4 isoform
1 [Homo sa...
           11      1  gb|EAW77317.1|  solute carrier family 30
(zinc transpor...
```

В наступному прикладі здійснюється друк слайсів результатів вирівнювань:

```
# друк слайсу трьох перших вирівнювань
blast_slice = blast_qresult[:3]
print(f'blast_slice = \n {blast_slice}')
Результат роботи коду:
blast_slice =
 Program: blastp (2.12.0+)
  Query: WP_106001333.1 (297)
         magnetosome biogenesis CDF transporter MamB
[Magnetospirillum gryphi...
 Target: nr
   Hits: ----  -----  ----------------------------------------------
--------------
            #  # HSP  ID + description
         ----  -----  ----------------------------------------------
--------------
            0      1  gb|AAP83846.1|  chromosome 4 open reading
frame 1 [Homo...
            1      1  gb|AAH22981.1|  SLC30A9 protein, partial
[Homo sapiens]
            2      1  dbj|BAG61754.1|  unnamed protein product
[Homo sapiens]
```

Надрукуємо також id послідовностей, які мають значимі співпадіння з послідовністю-запитом:

```
from Bio import SearchIO
# друк співпадінь
# hits - атрибут пакету Bio.SearchIO
print(f'blast_qresult.hits = \n {blast_qresult.hits}')

Результат роботи коду:
blast_qresult.hits =
 [Hit(id='gb|AAP83846.1|',   query_id='WP_106001333.1',   1   hsps),
Hit(id='gb|AAH22981.1|',   query_id='WP_106001333.1',   1   hsps),
Hit(id='dbj|BAG61754.1|',   query_id='WP_106001333.1',   1   hsps),
Hit(id='dbj|BAG35692.1|',   query_id='WP_106001333.1',   1   hsps),
Hit(id='gb|AAB87763.2|',   query_id='WP_106001333.1',   1   hsps),
Hit(id='ref|XP_016863143.1|',  query_id='WP_106001333.1',  1  hsps),
Hit(id='ref|NP_006336.3|',   query_id='WP_106001333.1',   1   hsps),
Hit(id='gb|AAY40966.1|',   query_id='WP_106001333.1',   1   hsps),
Hit(id='gb|EAW92998.1|',   query_id='WP_106001333.1',   1   hsps),
Hit(id='gb|AAB82561.1|',   query_id='WP_106001333.1',   1   hsps),
Hit(id='ref|NP_037441.2|',   query_id='WP_106001333.1',   1   hsps),
Hit(id='gb|EAW77317.1|',   query_id='WP_106001333.1',   1   hsps),
Hit(id='gb|ALQ33985.1|', query_id='WP_106001333.1', 1 hsps)]
```

Надрукуємо id і довжин послідовностей для перших п'яти вирівнювань:

```
# друк id і довжин послідовностей  для перших п'яти вирівнювань
for hit in blast_qresult[:5]:
    print(f'hit.id = {hit.id}, hit.seq_len = {hit.seq_len}', )

Результат роботи коду:
```

```
hit.id = gb|AAP83846.1|, hit.seq_len = 543
hit.id = gb|AAH22981.1|, hit.seq_len = 413
hit.id = dbj|BAG61754.1|, hit.seq_len = 397
hit.id = dbj|BAG35692.1|, hit.seq_len = 568
hit.id = gb|AAB87763.2|, hit.seq_len = 568
```

Пара сегментів з високою вагою (High-scoring Segment Pair - HSP) - це локальне вирівнювання, яке досягає однієї з найвищих ваг вирівнювання у даному пошуку. Надрукуємо слайси в парах сегментів з високою вагою:

```
# Слайси в парах сегментів з високою вагою
# blast_qresult = SearchIO.read("my_blast.xml", "blast-xml")
# перший [0] - це перше вирівнювання послідовності query,
# другий [0] - це перше вирівнювання з високою вагою послідовності
subject, тобто hsp
blast_hsp = blast_qresult[0][0]          # зріз (слайс) нульових
рядків query і subject
print(f'blast_hsp = \n {blast_hsp}')     # hsp - high score paire

Результат роботи коду:
blast_hsp =
       Query: WP_106001333.1 magnetosome biogenesis CDF
transporter MamB [Magn...
        Hit: gb|AAP83846.1| chromosome 4 open reading frame 1
[Homo sapiens]
Query range: [21:270] (0)
  Hit range: [218:495] (0)
Quick stats: evalue 9.2e-15; bitscore 77.41
  Fragments: 1 (286 columns)
     Query - ICMT----
LFKGILGLMSGSVALVADSLHSGADVVASGVTQLSLKISNKPADERYPFGY~~~EAVRQ
                      IC+      FK +  + +GS ++ ++++HS +D    G+  L +
S +  D  +P+G+    + +++
       Hit -
ICINGLNCFFKFLAWIYTGSASMFSEAIHSLSDTCNQGLLALGISKSVQTPDPSHPYGF~~~QEIQ
E
```

3.7. Використання Biopython для доступу до онлайн-ресурсів NCBI

Перш ніж використовувати Biopython для доступу до онлайн-ресурсів NCBI (через Bio.Entrez або деякі інші модулі), будь ласка, ознайомтеся з Вимогами користувача NCBI до Entrez https://www.ncbi.nlm.nih.gov/books/NBK25497/ [47].

Якщо NCBI виявить, що Ви зловживаєте їх системами, доступ буде заборонено!
Для будь-якої серії, що містить більше 100 запитів, робіть це у вихідні або за межами США пікових годин. Використовуйте адресу https://eutils.ncbi.nlm.nih.gov , а не стандартну вебадресу NCBI. Biopython використовує цю вебадресу.

3.8. Використання пакету Biopython для дослідження нуклеотидних послідовностей та білків на прикладі вірусу COVID-19

В роботі [48] (Nature Briefing) досліджено білки коронавірусу 2 під назвою важкий гострий респіраторний синдром SARS-CoV-2. Клоновано, визначено та експресовано 26 із 29 білків SARS-CoV-2 у клітинах людини та ідентифіковано білки людини, які фізично

асоціюються з кожним з білків SARS-CoV-2, з використанням методів мас-спектрометрії, визначено 332 високодостовірні білок-білкові взаємодії між SARS-CoV-2 та білками людини. Серед них виділено 66 білків людини або факторів-господарів, що піддаються медикаментозному впливу, орієнтовно на 69 сполук (з них 29 препаратів схвалено Управлінням з контролю за продуктами та ліками США, 12 знаходяться у клінічних, 28 – у доклінічних випробуваннях). В результаті проведених досліджень виявлено два набори фармакологічних засобів, які виявляли противірусну активність: інгібітори трансляції мРНК та передбачені регулятори рецепторів сигма-1 та сигма-2. Подальші дослідження цих агентів, спрямованих на фактор-хазяїна, включаючи їх комбінацію з препаратами, які безпосередньо спрямовані на вірусні ферменти, можуть призвести до терапевтичної схеми лікування COVID-19.

Рецептор сигма-1 (σ1R), один із двох підтипів сигма-рецепторів, є шаперонним білком в ендоплазматичному ретикулумі (ER), який модулює передачу сигналів кальцію через рецептор IP3 (рецептор трифосфат інозиту є специфічним лігандом кальцієвих каналів в мембрані гладкої ендоплазматичної мережі).

У людини рецептор σ1R кодується геном SIGMAR1. Рецептор σ1R являє собою трансмембранний білок експресується в багатьох типах тканин. Він особливо сконцентрований у певних областях центральної нервової системи. Рецептор σ1R залучений до кількох явищ, включаючи серцево-судинну функцію, шизофренію, клінічну депресію, ефекти зловживання кокаїном та рак. Багато відомо про спорідненість зв'язування сотень синтетичних сполук з рецептором σ1R.

Рецептор сигма-2 (σ2R) являє собою підтип сигма-рецепторів, який привернув увагу через його участь у таких захворюваннях, як рак та неврологічні захворювання. В даний час досліджується його можливе діагностичне та терапевтичне застосування.

При занесенні в NCBI послідовність вірусу SARS_COV_2, як і інших вірусів, зберігається в коді ACGT. Модуль SeqIO забезпечує простий уніфікований інтерфейс для введення та виведення різноманітних форматів файлів послідовності (включаючи вирівнювання кількох послідовностей), але працює лише з послідовностями (об’єктами) класу SeqRecord. Існує дочірній інтерфейс Bio.AlignIO для безпосередньої роботи з файлами вирівнювання послідовності як об’єктами вирівнювання. Розглянемо приклад використання Biopython для дослідження вірусу SARS_COV_2:

```
# Аналіз послідовності ДНК COVID-19.
import pandas as pd
path = "BioSeqExamples\SARS_COV_2_AA.fasta"
# Зчитування методом read() тільки послідовності з файлу без
атрибутів
from Bio import SeqIO
# SeqIO – модуль, read() повертає екземпляр класу SeqRecord
DNAsequence = SeqIO.read(path, "fasta")
print(DNAsequence)

Результат роботи коду:
ID: NC_045512.2
Name: NC_045512.2
Description: NC_045512.2 Severe acute respiratory syndrome
coronavirus 2 isolate Wuhan-Hu-1, complete genome
Number of features: 0
Seq('ATTAAAGGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGT...AAA'
)
```

Друк даних про послідовність у такому форматі відповідає класу SeqRecord.

Severe acute respiratory syndrome coronavirus 2 isolate Wuhan-Hu-1, complete genome – Важкий гострий респіраторний синдром коронавірус 2 ізольований з Ухань-Ху-1.
Наступний код здійснює друк інформації про послідовність вірусу SARS_COV_2:

```
#  Друк ID, кількості особливостей, самої послідовність вірусу
SARS_COV_2,
# представлення послідовності та її довжини
print(f"{DNAsequence.id} with {len(DNAsequence.features)}
features")
print(f'DNAsequence.id = {DNAsequence.id}')
print(f'repr(DNAsequence.seq) =\n {repr(DNAsequence.seq)}')
print(f'DNAsequence.seq = \n {DNAsequence.seq}')
print(f'len(DNAsequence) = {len(DNAsequence)} nucliotides')
```

В попередньому прикладі .id, .seq, .name, .features тощо – це атрибути класу SeqRecord.

```
Результат роботи коду:
NC_045512.2 with 0 features
DNAsequence.id = NC_045512.2
repr(DNAsequence.seq) =
Seq('ATTAAAGGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGT...AAA’
)
len(DNAsequence) = 29901 nucliotides
```

В наступному коді здійснюється транскрипція та трансляція послідовності вірусу:

```
# В змінній DNA зберігаємо послідовність
DNA = DNAsequence.seq
# Транскрипція ДНК в РНК - процес синтезу РНК з використанням ДНК
як матриці, тобто
# конвертуємо ДНК в РНК, заміна 'T' на 'U'
mRNA = DNA.transcribe()
# друкуємо РНК-послідовність, її довжину
print(mRNA)
print('Size : ', len(mRNA))
# Трансляція згідно табл 1(coding domain sequence ) cds=False -
перевірка наявності цілого числа кодонів
Amino_Acid = mRNA.translate(table=1, cds=False)
print('Amino Acid', Amino_Acid)
# друкуємо кількість амінокислотних залишків після трансляції
print("Length of Protein:", len(Amino_Acid))
# для порівняння ще раз друкуємо довжину РНК послідовності
print("Length of Original mRNA:", len(mRNA))

Результат роботи коду:
len(DNAsequence) = 29901 nucliotides
AUUAAAGGUUUAUACCUUCCCAGGUAACAAACCAACCAACUUU…….
Size :  29901
Amino Acid IKGLYLPR*QTNQLSISCRSVL*TNFKICVAVTRLHA*CTHAV*LITNYCR*……
Length of Protein: 9967
Length of Original mRNA: 29901
```

Якщо в цьому коді задати cds = True, Biopython інтерпретує це як повний coding domain sequence (послідовність, що кодує домен) – CDS, тобто Biopython перевіряє наявність цілого

числа кодонів (послідовність кратна трьом за довжиною), перевіряє, що останній кодон є стоп-кодоном тощо.

Виведемо на друк стандартну таблицю генетичного коду (Рисунок 3.25):

```
# Друкуємо таблицю 1- 'Standard'
from Bio.Data import CodonTable
print(CodonTable.unambiguous_rna_by_name['Standard'])
```

В модулі CodonTable пакету Bio.Data unambiguous_rna_by_name – словник, в якому ключ – це назва таблиці або її номер, а значення – це сама таблиця.

Наступний код формує інформацію про білки:

```
# Ідентифікуємо всі білки,  * - це стоп-кодони
Amino_Acid = mRNA.translate(table=1, cds=False)
Proteins = Amino_Acid.split('*')                # '*' - роздільник рядків
# print(f'Proteins = \n {Proteins}')
df = pd.DataFrame(Proteins)
df.describe()          # функція describe() pandas.DataFrame для створення описової статистики
print(f'df = \ {df}')
print(f'df.describe() = \n {df.describe()}')
df.head()      # функція head() pandas.DataFrame повертає перші n рядків для об’єкта
```

```
Table 1 Standard, SGC0

  |  U       |  C       |  A       |  G       |
--+----------+----------+----------+----------+--
U | UUU F    | UCU S    | UAU Y    | UGU C    | U
U | UUC F    | UCC S    | UAC Y    | UGC C    | C
U | UUA L    | UCA S    | UAA Stop | UGA Stop | A
U | UUG L(s) | UCG S    | UAG Stop | UGG W    | G
--+----------+----------+----------+----------+--
C | CUU L    | CCU P    | CAU H    | CGU R    | U
C | CUC L    | CCC P    | CAC H    | CGC R    | C
C | CUA L    | CCA P    | CAA Q    | CGA R    | A
C | CUG L(s) | CCG P    | CAG Q    | CGG R    | G
--+----------+----------+----------+----------+--
A | AUU I    | ACU T    | AAU N    | AGU S    | U
A | AUC I    | ACC T    | AAC N    | AGC S    | C
A | AUA I    | ACA T    | AAA K    | AGA R    | A
A | AUG M(s) | ACG T    | AAG K    | AGG R    | G
--+----------+----------+----------+----------+--
G | GUU V    | GCU A    | GAU D    | GGU G    | U
G | GUC V    | GCC A    | GAC D    | GGC G    | C
G | GUA V    | GCA A    | GAA E    | GGA G    | A
G | GUG V    | GCG A    | GAG E    | GGG G    | G
--+----------+----------+----------+----------+--
```

Рисунок 3.25 – Стандартна таблиця генетичного коду.

В цьому коді використовуються такі методи:

- .split() – метод розбиває рядок на список підрядків. Можна вказати будь-який роздільник, роздільником за замовчуванням є пробіл, в цьому прикладі «*».

- .head() – це функція pandas.DataFrame, яка повертає перші n рядків для об’єкта. Це корисно для швидкого тестування, для перевірки правильності типу даних об'єкту.
- .describe() – функція pandas.DataFrame створення описової статистики. Для числових даних результат включає count, mean, std (standard deviation – середньоквадратичне відхилення), min, max тощо. Для рядків результат включає count, unique, top, top – це найбільш поширене значення, freq – це частота найпоширенішого значення.

```
Результат роботи коду:
Proteins =
 [Seq('IKGLYLPR'), Seq('QTNQLSISCRSVL'), Seq('TNFKICVAVTRLHA'),
Seq('CTHAV'), Seq('LITNYCR’), ……
df =
0                                (I, K, G, L, Y, L, P, R)
1                  (Q, T, N, Q, L, S, I, S, C, R, S, V, L)
2              (T, N, F, K, I, C, V, A, V, T, R, L, H, A)
3                                        (C, T, H, A, V)
4                                  (L, I, T, N, Y, C, R)
..                                                    ...
770          (S, H, I, A, I, F, N, Q, C, V, T, L, G, R, T)
771   (K, S, H, H, I, F, T, E, A, T, R, S, T, I, E, ...
772                                                  (F)
773                                                   ()
774    (L, L, R, R, M, T, K, K, K, K, K, K, K, K, K, K)
[775 rows x 1 columns]
df.describe() =
                    0
count             775
unique            625
top                ()
freq               69
df.head() =
                                                       0
0                      (I, K, G, L, Y, L, P, R)
1       (Q, T, N, Q, L, S, I, S, C, R, S, V, L)
2    (T, N, F, K, I, C, V, A, V, T, R, L, H, A)
3                               (C, T, H, A, V)
4                         (L, I, T, N, Y, C, R)
Total proteins: 775
```

Сформуємо таблицю даних про білки:

```
# Формування таблиці df
def conv(item):   # функція, яка повертає довжину об'єкта
    return len(item)
def to_str(item):   # функція, яка перетворює об'єкт на рядок
    return str(item)
df['sequence_str'] = df[0].apply(to_str) # функція apply(to_str)
переводить кортеж символів в рядок
                                         # та формується новий
стовпчик таблиці даних 'sequence_str'
print('sequence_str', df[0].apply(to_str))  # apply() працює
тільки з об’єктами pandas.DataFrame
```

```
print(f'df.head() = \n {df.head()}')
print('Total proteins:', len(df))
```

Однією з альтернатив використання циклу для ітерації в DataFrame є використання методу .apply() pandas.DataFrame. Ця функція діє як функція map() в python. Вона приймає функцію як вхід і застосовує цю функцію до всієї маркованої таблиці даних DataFrame. Якщо Ви працюєте з табличними даними, Ви повинні вказати вісь, на якій Ваша функція буде діяти (0 - для стовпців; 1 - для рядків).

Подібно до функції map(), метод apply() використовує функції в якості аргументів.

```
Результат роботи коду:
 sequence_str
0                                          IKGLYLPR
1                                       QTNQLSISCRSVL
2                                      TNFKICVAVTRLHA
3                                               CTHAV
4                                            LITNYCR
                            ...
770                                     SHIAIFNQCVTLGRT
771    KSHHIFTEATRSTIECTVNNARESCLYGRALMCKINFSSAIPM
772                                                F
773
774                                    LLRRMTKKKKKKKKK
Name: 0, Length: 775, dtype: object
```

Розрахуємо довжини білків та сформуємо додаткові стовпчики таблиці результатів:

```
# Розрахунок довжини і додавання стовпчика 'length' до таблиці даних
df['length'] = df[0].apply(conv)     # apply(conv) функція, яка повертає довжину рядка,
                                     # створеного функцією conv() на попередньому слайді та
                                     # формується новий стовпчик таблиці даних 'length'
prot_len = df[0].apply(conv)        # метод apply() використовує функцію conv() в якості аргументу
print('length = \n', prot_len)
# формування додаткових стовпчиків таблиці
df.rename(columns={0: "sequence"}, inplace=True) # .rename змінює мітки, тобто маркери стовпчиків
# inplace за замовчуванням False. True означає, що повертається нова таблиця даних DataFrame.
# функція head() повертає перші n рядків таблиці даних
df.head()
print('df\n', df.head())
```

```
Результат роботи коду:
length of proteins =
 0       8
1      13
2      14
3       5
```

```
4       7
       ..
770    15
771    43
772     1
773     0
16
Name: 0, Length: 775, dtype: int64
df
                                  sequence
sequence_str                       length
0                    (I, K, G, L, Y, L, P, R)
IKGLYLPR                                8
1      (Q, T, N, Q, L, S, I, S, C, R, S, V, L)
QTNQLSISCRSVL                          13
2   (T, N, F, K, I, C, V, A, V, T, R, L, H, A)        TNFKICVAVTRLHA
14
3                              (C, T, H, A, V)
CTHAV                                   5
4                       (L, I, T, N, Y, C, R)
LITNYCR                                 7
```

Здійснимо відбір білків з довжинами, що перевищують задане значення:

```
# Відбираємо білки з довжиною більше 40
functional_proteins = df.loc[df['length'] >= 40]
# функція len() повертає кількість білків, довжина яких >= 40
print('Total functional proteins:', len(functional_proteins))
# функція describe pandas.DataFrame для створення описової
статистики
functional_proteins.describe()
print(functional_proteins)
```

Властивість .loc[] pandas.DataFrame надає доступ до групи рядків і стовпчиків за мітками або за логічним масивом.

```
Результат роботи коду:
Total functional proteins: 13
functional_proteins =       sequence  ... … …      length
6       (D, G, E, P, C, P, W, F, Q, R, E, N, T, R, P, ...  ...
46
242   (R, N, F, V, L, H, R, R, C, F, T, Y, K, V, L, ...  ...
40
464   (T, M, L, R, C, Y, F, P, K, C, S, E, K, N, N, ...  ...     46
539   (D, V, V, Y, T, H, W, Y, W, S, G, N, N, S, Y, ...  ...    43
548   (C, T, I, V, F, K, R, V, C, G, V, S, A, A, R, ...  ...   2701
674   (F, L, W, K, G, L, S, S, Y, V, L, P, S, V, S, ...  ...
42
694   (A, S, A, Q, R, S, Q, I, T, L, H, I, N, E, L, ...  ...    290
695   (A, Q, A, D, E, Y, E, L, M, Y, S, F, V, S, E, ...  ...
83
718   (Q, Q, M, F, H, L, V, D, F, Q, V, T, I, A, E, ...  ...     63
```

```
719  (T, N, M, K, I, I, L, F, L, A, L, I, T, L, A, ...  ...
123
729  (L, N, C, A, W, M, R, L, V, L, N, H, P, F, S, ...  ...     41
758  (L, Q, T, L, A, A, N, C, T, I, C, P, Q, R, F, ...  ...
43
771  (K, S, H, H, I, F, T, E, A, T, R, S, T, I, E, ...  ...
43
```

Здійснимо перетворення кортежу в рядок:

```
# Переводимо кортеж стовпчика 'sequence' в рядок, довжина білка >=
40
funct_prot = list(map(str, functional_proteins['sequence']))
print(f'funct_prot = {funct_prot}')

Результат роботи коду:
funct_prot =
['DGEPCPWFQRENTRPTQFACFTGSRRARTWLWRLRGGGLIRGTSTS',
'RNFVLHRRCFTYKVLRIQRSYYGCFLQRKQLHNNHKTSYL',
'TMLRCYFPKCSEKNNQGYTPLVVTHNFDFTFSFSPEYSMVFVLFFV', ...    …
'ASAQRSQITLHINELMDLFMRIFTIGTVTLKQGEIKDATPSDFVRATATIPIQASLPFGWLIVGV
ALLAVFQSASKIITLKKRWQLALSKGVHFVCNLLLLFVTVYSHLLLVAAGLEAPFLYLYALVYFLQ
SINFVRIIMRLWLCWKCRSKNPLLYDANYFLCWHTNCYDYCIPYNSVTSSIVITSGDGTTSPISEH
DYQIGGYTEKWESGVKDCVVLHSYFTSDYYQLYSTQLSTDTGVEHVTFFIYNKIVDEPEEHVQIHT
IDGSSGVVNPVMEPIYDEPTTTTSVPL',
'AQADEYELMYSFVSEETGTLIVNSVLLFLAFVVFLLVTLAILTALRLCAYCCNIVNVSLVKPSFY
VYSRVKNLNSSRVPDLLV',
'QQMFHLVDFQVTIAEILLIIMRTFKVSIWNLDYIINLIIKNLSKSLTENKYSQLDEEQPMEID',
'TNMKIILFLALITLATCELYHYQECVRGTTVLLKEPCSSGTYEGNSPFHPLADNKFALTCFSTQF
AFACPDGVKHVYQLRARSVSPKLFIRQEEVQELYSPIFLIVAAIVFITLCFTLKRKTE']
```

3.9. Аналіз білкових послідовностей за допомогою модуля ProtParam

Білкові послідовності можна аналізувати кількома методами на основі класів модуля ProtParam на сервері Expasy (Expert Protein Analysis System)) Proteomics. У червні 2011 році сервер Expasy став SIB ExPASy Bioformatics Resources Portal: різноманітний каталог біоінформаційних ресурсів, розроблений SIB Groups. Поточна версія Expasy була випущена в жовтні 2020 року. SIB – Swiss Institute of Bioinformatics.
Модуль ProtParam є частиною пакета SeqUtils на сервері Expasy.
Клас ProteinAnalysis приймає один аргумент, послідовності білка у вигляді рядка і будує об'єкт за допомогою модуля Bio.Seq.

ProtParam – це інструмент, який дозволяє обчислювати різні фізичні та хімічні параметри для даного білка, що зберігається в Swiss-Prot або TrEMBL або для послідовності білків, яка введена користувачем. Розраховані параметри включають амінокислотний склад, частку кожної амінокислоти, молекулярну масу, параметри ароматичності, індексів нестабільності, теоретичні значення ізоелектричних точок pI, типи вторинних структур, оцінку періоду напіврозпаду, індекс нестабільності, аліфатичний індекс для кожного білка.

Наступний код є прикладом розрахунку характеристик білків:

```
# Імпортуємо модуль ProtParam із пакету Bio.SeqUtils
# для дослідження параметрів білків
from Bio.SeqUtils import ProtParam
# формуємо порожні списки для подальшого зберігання параметрів
білків
```

```
poi_list = []                # список кількості кожної з 20
амінокислот в кожному білку
per = dict()                 # частка кожної амінокислоти
per_list = []                # список частки кожної амінокислоти  в
кожному білку
MW_list = []                 # список молекулярних ваг в кожному
білку
arom_list = []               # список параметрів ароматичності в
кожному білку
flex_list = []               # список індексів нестабільності для
кожного білка
isoel_list = []              # список теоретичних значень
ізоелектричних точок для
                             # кожного білка
sec_struct_list = []         # список типів вторинних структур для
кожного білка
instab_list = []             # список індексів нестабільності білків
```

З застосуванням модуля ProtParam із пакету Bio.SeqUtils можна здійснити розрахунок таких характеристик білків (https://biopython.org/wiki/ProtParam [49]):

- count_amino_acids() – розрахунок кількості кожної з 20 амінокислот в білку. Повертає словник {AminoAcid: Number}, а також зберігає словник.
- get_amino_acids_percent() – розрахунок частки кожної з 20 амінокислот в білку в частка від одиниці. Повертає словник, а також зберігає словник.
- molecular_weight() – розрахунок молекулярної ваги - маси молекули, вираженої в атомних одиницях маси. Атомна одиниця маси (а.о.м., інша назва дальтон) - позасистемна одиниця маси, з 1960 року встановлена як 1/12 маси ізотопу вуглецю ^{12}C з масовим числом 12 (так звана *вуглецева шкала*). 1 а.о.м = 1 u ≈ 1,660538782(83) × 10^{-27} кг
- count_amino_acids() – розрахунок кількості кожної з 20 амінокислот в білку. Повертає словник {AminoAcid: Number}, а також зберігає словник.
- get_amino_acids_percent() – розрахунок частки кожної з 20 амінокислот в білку в частка від одиниці/ Повертає словник, а також зберігає словник.
- molecular_weight() – розрахунок молекулярної ваги - маси молекули, вираженої в атомних одиницях маси. Атомна одиниця маси (а.о.м., інша назва дальтон) позасистемна одиниця маси, з 1960 року встановлена як 1/12 маси ізотопу вуглецю ^{12}C з масовим числом 12 (так звана *вуглецева шкала*). 1 а.о.м = 1 u ≈ 1,660538782(83) × 10^{-27} кг

Розрахунок характеристик білків здійснюється в наступному коді:

```
from Bio.SeqUtils import ProtParam
# Розрахунок параметрів білків
for record in funct_prot:
    print(f'record = {record}')
    # Створюємо екземпляр X класу ProteinAnalysis
    X = ProtParam.ProteinAnalysis(record)
    poi = X.count_amino_acids()
    poi_list.append(poi)
    per = X.get_amino_acids_percent()
    per_list.append(X.get_amino_acids_percent())
    MW = X.molecular_weight()
    MW_list.append(MW)
    arom = X.aromaticity()
```

```
    arom_list.append(arom)
    flex = X.flexibility()
    flex_list.append(flex)
    isoel = X.isoelectric_point()
    isoel_list.append(isoel)
    sec_struct = X.secondary_structure_fraction()
    sec_struct_list.append(sec_struct)
for record in Proteins[:]:          # Proteins =
Amino_Acid.split('*')
    print("\n")                                  # клас ProteinAnalysis
    X = ProtParam.ProteinAnalysis(str(record))
    poi = X.count_amino_acids()
    poi_list.append(poi)
    try:
        per = X.get_amino_acids_percent()
        per_list.append(X.get_amino_acids_percent())
    except ZeroDivisionError:
        per_list.append({'EmptyError': 'Empty Sequence'})
    MW = X.molecular_weight()
    MW_list.append(MW)
    try:
        arom = X.aromaticity()
        arom_list.append(arom)
    except ZeroDivisionError:
        arom_list.append('Empty Sequence')
        flex = X.flexibility()
        flex_list.append(flex)
    try:
        isoel = X.isoelectric_point()
        isoel_list.append(isoel)
    except IndexError:
        isoel_list.append('Empty Sequence')
    try:
        sec_struct = X.secondary_structure_fraction()
        sec_struct_list.append(sec_struct)
    except ZeroDivisionError:
        sec_struct_list.append('Empty Sequence')
```

Також з застосуванням модуля ProtParam із пакету Bio.SeqUtils можна здійснити розрахунок ароматичності як характеристики білка – aromaticity https://biopython.org/wiki/ProtParam [49].

Ароматичні сполуки – це ненасичені циклічні та плоскі молекули, які містять ароматичне кільце. Вони мають додаткову стійкість в результаті розташування π -електронів, розташованих вище і нижче площини ароматичного кільця. Ці електрони породжують так звану хмару π-електронів над кільцем. Ароматичність – це хімічна властивість, пов'язана з такими циклічними та площинними сполуками, і приписується цим π-електронам, які можуть вільно кружляти навколо кругового розташування атомів, що знаходяться в ароматичних частинах. Це явище можна розглядати як прояв циклічної делокалізації, яка зустрічається в планарних кільцевих системах, таких як бензол. Плоска грань ароматичного кільця має частковий негативний заряд завдяки цим π -електронам.

Тирозин (Tyr), Фенілаланін (Phe) і Триптофан (Trp) - це три ароматичні амінокислоти (ААА), які беруть участь у синтезі білка. Ці амінокислоти та їх метаболізм пов’язані з

синтезом різноманітних вторинних метаболітів, підмножина яких бере участь у численних анаболічних шляхах, що відповідають за синтез пігментних сполук, гормонів рослин та біологічних полімерів. Крім того, ці метаболіти, отримані з шляхів AAA, опосередковують передачу нервових сигналів, нейтралізують активні форми кисню в мозку та беруть участь у забарвленні тварин. Організми, які не мають ферментативного механізму для синтезу AAA, повинні отримувати ці первинні метаболіти зі свого раціону.

Характеристика білка aromaticity() – це відносна частота зустрічаємості ароматичних амінокислот: Phe (Фенілаланін)+Trp (Триптофан) +Tyr (Тирозин), розраховується за методом [50]. Однак, тирозин не тільки ароматична, а і полярна амінокислота; а гістидин хоч і містить ароматичне кільце, згідно з його властивостями класифікується як полярна амінокислота. Взаємодії, що відбуваються між бічними ланцюгами залишків ароматичних амінокислот, називаються ароматично-ароматичними взаємодіями. Ароматичні взаємодії відносно неполярні. Виявлено, що вони відіграють важливу роль у підтримці загальної структури білкових молекул та комплексів білок-ДНК. Взаємодія між ароматичними залишками всередині білка, а також у комплексі білок-ДНК є невід'ємною частиною для належного функціонування молекули білка [50].

Характеристика білка flexibility() – це гнучкість білка за методом [51]. Структурна гнучкість білка важлива для каталізу та зв'язування. Параметри гнучкості білка обернено корелюють зі стабільністю білка.

Організація структури білка характеризується конформаційним розташуванням повторюваних структур (вторинні структури, тобто α-спіралі, β-листи та петлі). Спостереження за організацією білків виявило деякі їх основні властивості, тобто активні центри зазвичай знаходяться в ядрі білка, в якому амінокислотні залишки добре упаковані і переважно гідрофобні, тоді як поверхневі амінокислотні залишки, піддаються впливу розчинника або інших факторів (білок, ДНК), є більш гнучкими, оскільки менш обмежені, ніж білки ядра. Функції білків та механізми їх взаємодії потребують наявності деяких гнучких властивостей, які значно складніші за ці спрощені уявлення. З використанням різних джерел структурних даних та розробкою різних обчислювальних методів виявилося, що динаміка білків охоплює великий спектр конформаційних змін (у поєднанні з рухливістю жорсткого фрагмента та деформацією білкового каркасу), включаючи наявність внутрішньої невпорядкованої області тощо. Гнучка структура може дозволити білку зв'язуватися з багатьма партнерами [52], низька спорідненість до зв'язування з багатьма партнерами може характеризуватися високою специфічністю білка [53].

Гнучкість є важливою для виконання функцій білків. Аналіз білкових структур, їх динаміки, достатньо точний і повний опис конформації білкового каркасу можна отримати за допомогою бібліотек невеликих фрагментів білка, які називаються структурними алфавітами Structural alphabets (SAs) та використовуються в області аналізу структур білків, динаміки білкових структур, визначенню місць зв'язування лігандів тощо. У поєднанні з різними джерелами експеримент. даних (наприклад, B-фактор) і обчислювальною методологією (наприклад, молекулярно-динамічне моделювання), SA виявляються потужними інструментами для аналізу динаміки білків, наприклад, для дослідження алостеричних механізмів (алостеричний ефект – це зміна поведінки в одній частині молекули, викликана зміною в іншій її частині) у великому наборі структур у комплексах, щоб визначити конформаційні переходи.

Щоб краще оцінити гнучкість амінокислот, в роботі [54] досліджено B-фактори (відображають ступінь теплового руху) з великого набору кристалічних структур з високою роздільною здатністю, враховували B-фактори для кожної амінокислоти в досліджуваному наборі. Було виявлено, що розподіл B-факторів (розподілом Гумбеля) (узагальнений розподіл екстремальних значень). Режим цього розподілу використовувався як оцінка гнучкості для амінокислот. Розподіл Гумбеля є розподілом екстремальних значень як максимальних так і мінімальних вибірок. Використовується для моделювання розподілу

пікових рівнів. Наприклад, показує розподіл пікових температур протягом року, якщо існує список максимальних температур протягом 10 років.

На основі режиму розподілу Гумбеля була отримана оцінка гнучкості кожної амінокислоти. Амінокислоти, які зазвичай мають менші B-фактори, матимуть менші режими (іншими словами параметри розташування).

Порядок амінокислот за висхідним параметром розташування був WYFCIVHLMAGTRSNQDPEK. Виходячи з цих параметрів розташування, амінокислоти були поділені на дві групи по 10, де WYFCIVHLMA визначено як жорсткі амінокислоти, а GTRSNQDPEK визначено як гнучкі амінокислоти (Таблиця 3.2).

Таблиця 3.2. Гнучкість амінокислот [55].

W (Trp): 0.310
Y (Tyr): 0.420
F (Phe): 0.310
C (Cys): 0.350
I (Ile): 0.460
V (Val): 0.390
H (His): 0.320
L (Leu): 0.370
M (Met): 0.300
A (Ala): 0.360
G (Gly): 0.540
T (Thr): 0.440
R (Arg): 0.530
S (Ser): 0.510
N (Asn): 0.460
Q (Gln): 0.490
D (Asp): 0.510
P (Pro): 0.510
E (Glu): 0.500
K (Lys): 0.470

Також з застосуванням модуля ProtParam із пакету Bio.SeqUtils можна здійснити розрахунок ізоелектричної точки (pI) білків. Ізоелектрична точка (pI) isoelectric_point() – це кислотність середовища pH, за якого молекула певного білка не несе електричного заряду (стає електронейтральною).

Чим більше в цьому білку гідроксильних груп (основних залишків), тим вища в нього pI.

Білки з pI, меншим за 7, називаються кислотними, а білки з pI, більшим за 7, – основними.

Загалом pI білка залежить від функції, яку він виконує. Так, білки, що зв’язуються з нуклеїновими кислотами, часто належать до основних білків. Прикладом таких білків служать гістони.

Також з застосуванням модуля ProtParam із пакету Bio.SeqUtils можна здійснити розрахунок фракцій вторинної структури білків. Для цього використовується метод secondary_structure_fraction() - фракція вторинної структури. Цей метод повертає список фракцій амінокислот, які, як правило, знаходяться у спіралі, повороті або листі (Рисунок 3.26):

- Амінокислоти у спіралі (helix): V, I, Y, F, W, L.
- Амінокислоти в повороті (turn): N, P, G, S. Поворот - це елемент вторинної структури білків, де поліпептидний ланцюг змінює свій загальний напрямок.
- Амінокислоти в листі (sheet): E, M, A, L.
- Список містить 3 значення: [Helix, Turn, Sheet].

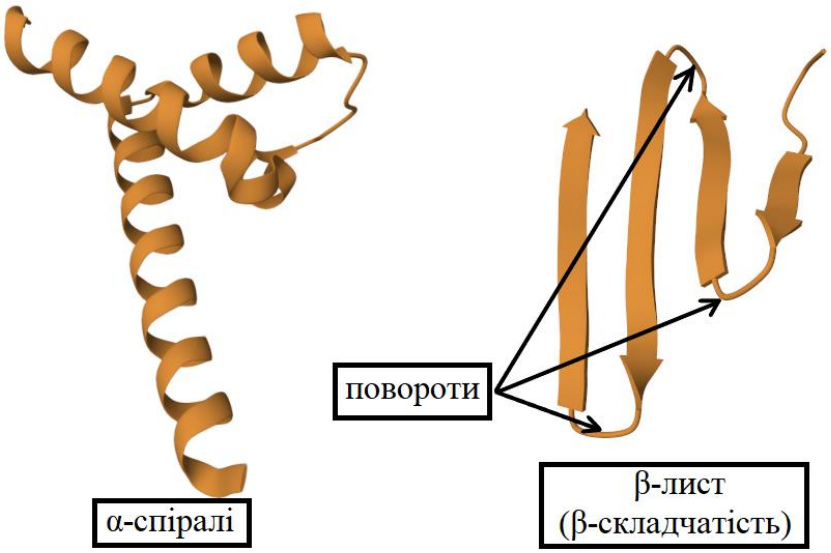


Рисунок 3.26 – Змодельований вигляд α-спіралі, повороту та β-листа.

Також з застосуванням модуля ProtParam із пакету Bio.SeqUtils можна здійснити розрахунок індексу стабільності білків. Для цього використовується метод instability_index() – розрахунок стабільності білка. Даний метод реалізовано згідно методу [56]. Цей метод перевіряє стабільність білка. Будь-яке значення вище 40 означає, що білок нестійкий (має короткий період напіввиведення).

За класифікацією [57], набір з 32 білків з періодом напіврозпаду in vivo > 16 годин брали за клас стабільних білків і набір з 12 білків з періодом напіввиведення in vivo <5 годин вибирали як нестійкий клас білків. Послідовності з цих двох наборів білків аналізували окремо. Частота появи кожної з 20 амінокислот була розрахована у нестабільному, а також стабільному класі білків і порівнювалася з частотою виникнення різних амінокислот у загальній базі даних послідовностей білків PIR. Ці дані явно показують певні помітні відмінності між частотою виникнення різних амінокислот в нестійких і стабільних білках. У разі нестабільних білків амінокислоти Met (M), Gln (Q), Pro (P), Glu (E) та Ser (S) зустрічаються з відносно високою частотою. Гіпотеза PEST, запропонована раніше Роджерсом та ін. (1986) повідомляє про наявність області, що складається з амінокислот (Pro) P, (Glu) E, (Ser) S і (Thr) T у нестійких білках. Існує також гіпотеза, що Thr (T) не зустрічається більш часто в нестабільних білках порівняно зі стабільними білками, а така амінокислота, як Met (M), більш часто зустрічається в нестабільних білках.

Аналогічно в низці робіт було відзначено, що амінокислоти Asn (N), Lys (K) і Gly (G) зустрічаються з відносно вищими частотами в стабільних білках у порівнянні з нестабільними.

Гіпотеза PEST говорить про ділянки негативно заряджених амінокислот, які, як правило, групуються і, як правило, оточені позитивно зарядженими амінокислотами. Тому виявляється, що характеристики нестабільності або стійкості ймовірно, регулюються розташуванням певних амінокислот у певному порядку.

Звідси найменша одиниця, що визначає порядок в послідовності є дипептидом, поява амінокислоти у зіставленні з іншою може бути значним фактором визначення стабільності білка. Тому досліджено появу всіх 400 можливих дипептидів у двох класів білків [56].

Наступний код здійснює друк попередньо розрахованих характеристик білків:

```
print("Count of aminoacids in protein of interest = \n",
poi_list[:10])
print("type of XX.get_amino_acids_percent() = \n", per_list[:10])
```

```
print("Molecular weight = ", MW_list)
print("Aromaticity = \n", arom_list)
print("Flexibility = ", flex_list[0])
print("Isoelectric point = ", isoel_list)
print("Secondary structure fraction = ", sec_struct_list)
print("Instability index = ", instab_list)
```

Результати роботи коду з розрахунку кількості кожної з амінокислот в білку:

```
Count of aminoacids in protein of interest =
[{'A': 2, 'C': 2, 'D': 1, 'E': 2, 'F': 3, 'G': 6, 'H': 0, 'I': 1,
'K': 0, 'L': 3, 'M': 0, 'N': 1, 'P': 3, 'Q': 2, 'R': 8, 'S': 3,
'T': 6, 'V': 0, 'W': 3, 'Y': 0}, {'A': 0, 'C': 2, 'D': 0, 'E': 0,
'F': 3, 'G': 1, 'H': 3, 'I': 1, 'K': 3, 'L': 5, 'M': 0, 'N': 3,
'P': 0, 'Q': 3, 'R': 6, 'S': 2, 'T': 2, 'V': 2, 'W': 0, 'Y': 4},
{'A': 0, 'C': 2, 'D': 1, 'E': 2, 'F': 8, 'G': 1, 'H': 1, 'I': 0,
'K': 2, 'L': 3, 'M': 2, 'N': 3, 'P': 3, 'Q': 1, 'R': 1, 'S': 4,
'T': 4, 'V': 5, 'W': 0, 'Y': 3}, {'A': 0, 'C': 1, 'D': 1, 'E': 0,
'F': 0, 'G': 3, 'H': 2, 'I': 2, 'K': 1, 'L': 4, 'M': 0, 'N': 2,
'P': 2, 'Q': 1, 'R': 2, 'S': 8, 'T': 2, 'V': 4, 'W': 4, 'Y': 4},
{'A': 179, 'C': 89, 'D': 178, 'E': 101, 'F': 143, 'G': 149, 'H':
70, 'I': 129, 'K': 159, 'L': 233, 'M': 63, 'N': 151, 'P': 110,
'Q': 88, 'R': 113, 'S': 162, 'T': 183, 'V': 229, 'W': 32, 'Y':
140}, {'A': 1, 'C': 6, 'D': 1, 'E': 1, 'F': 1, 'G': 1, 'H': 2,
'I': 0, 'K': 2, 'L': 8, 'M': 0, 'N': 1, 'P': 2, 'Q': 0, 'R': 1,
'S': 8, 'T': 2, 'V': 2, 'W': 2, 'Y': 1}, {'A': 15, 'C': 7, 'D':
13, 'E': 12, 'F': 14, 'G': 14, 'H': 9, 'I': 23, 'K': 11, 'L': 32,
'M': 4, 'N': 9, 'P': 12, 'Q': 11, 'R': 7, 'S': 24, 'T': 25, 'V':
25, 'W': 6, 'Y': 17}, {'A': 6, 'C': 3, 'D': 2, 'E': 4, 'F': 5,
'G': 1, 'H': 0, 'I': 3, 'K': 2, 'L': 15, 'M': 1, 'N': 5, 'P': 2,
'Q': 1, 'R': 3, 'S': 8, 'T': 4, 'V': 13, 'W': 0, 'Y': 5}, {'A': 1,
'C': 0, 'D': 4, 'E': 5, 'F': 3, 'G': 0, 'H': 1, 'I': 10, 'K': 4,
'L': 8, 'M': 3, 'N': 4, 'P': 1, 'Q': 5, 'R': 1, 'S': 4, 'T': 3,
'V': 3, 'W': 1, 'Y': 2}, {'A': 9, 'C': 6, 'D': 2, 'E': 8, 'F': 10,
'G': 4, 'H': 3, 'I': 8, 'K': 7, 'L': 15, 'M': 1, 'N': 3, 'P': 6,
'Q': 5, 'R': 5, 'S': 7, 'T': 11, 'V': 8, 'W': 0, 'Y': 5}]
```

Результати роботи коду з розрахунку частки кожної з амінокислот в білку:

```
type of XX.get_amino_acids_percent() =
 [{'A': 0.043478260869565216, 'C': 0.043478260869565216, 'D':
0.021739130434782608, 'E': 0.043478260869565216, 'F':
0.06521739130434782, 'G': 0.13043478260869565, 'H': 0.0, 'I':
0.021739130434782608, 'K': 0.0, 'L': 0.06521739130434782, 'M':
0.0, 'N': 0.021739130434782608, 'P': 0.06521739130434782, 'Q':
0.043478260869565216, 'R': 0.17391304347826086, 'S':
0.06521739130434782, 'T': 0.13043478260869565, 'V': 0.0, 'W':
0.06521739130434782, 'Y': 0.0}, {'A': 0.0, 'C': 0.05, 'D': 0.0,
'E': 0.0, 'F': 0.075, 'G': 0.025, 'H': 0.075, 'I': 0.025, 'K':
0.075, 'L': 0.125, 'M': 0.0, 'N': 0.075, 'P': 0.0, 'Q': 0.075,
'R': 0.15, 'S': 0.05, 'T': 0.05, 'V': 0.05, 'W': 0.0, 'Y': 0.1},
{'A': 0.0, 'C': 0.043478260869565216, 'D': 0.021739130434782608,
'E': 0.043478260869565216, 'F': 0.17391304347826086, 'G':
0.021739130434782608, 'H': 0.021739130434782608, 'I': 0.0, 'K':
0.043478260869565216, 'L': 0.06521739130434782, 'M':
```

0.043478260869565216, 'N': 0.06521739130434782, 'P':
0.06521739130434782, 'Q': 0.021739130434782608, 'R':
0.021739130434782608, 'S': 0.08695652173913043, 'T':
0.08695652173913043, 'V': 0.10869565217391304, 'W': 0.0, 'Y':
0.06521739130434782}, {'A': 0.0, 'C': 0.023255813953488372, 'D':
0.023255813953488372, 'E': 0.0, 'F': 0.0, 'G':
0.06976744186046512, 'H': 0.046511627906976744, 'I':
0.046511627906976744, 'K': 0.023255813953488372, 'L':
0.09302325581395349, 'M': 0.0, 'N': 0.046511627906976744, 'P':
0.046511627906976744, 'Q': 0.023255813953488372, 'R':
0.046511627906976744, 'S': 0.18604651162790697, 'T':
0.046511627906976744, 'V': 0.09302325581395349, 'W':
0.09302325581395349, 'Y': 0.09302325581395349}, {'A':
0.06627175120325805, 'C': 0.03295075897815624, 'D':
0.06590151795631248, 'E': 0.03739355794150315, 'F':
0.052943354313217325, 'G': 0.05516475379489078, 'H':
0.0259163272861903, 'I': 0.047760088855979266, 'K':
0.05886708626434654, 'L': 0.08626434653831914, 'M':
0.02332469455757127, 'N': 0.055905220288781934, 'P':
0.04072565716401333, 'Q': 0.03258052573121066, 'R':
0.041836356904850054, 'S': 0.05997778600518327, 'T':
0.06775268419104036, 'V': 0.08478341355053684, 'W':
0.011847463902258423, 'Y': 0.0518326545723806}, {'A':
0.023809523809523808, 'C': 0.14285714285714285, 'D':
0.023809523809523808, 'E': 0.023809523809523808, 'F':
0.023809523809523808, 'G': 0.023809523809523808, 'H':
0.047619047619047616, 'I': 0.0, 'K': 0.047619047619047616, 'L':
0.19047619047619047, 'M': 0.0, 'N': 0.023809523809523808, 'P':
0.047619047619047616, 'Q': 0.0, 'R': 0.023809523809523808, 'S':
0.19047619047619047, 'T': 0.047619047619047616, 'V':
0.047619047619047616, 'W': 0.047619047619047616, 'Y':
0.023809523809523808}, {'A': 0.05172413793103448, 'C':
0.02413793103448276, 'D': 0.04482758620689655, 'E':
0.041379310344827586, 'F': 0.04827586206896552, 'G':
0.04827586206896552, 'H': 0.03103448275862069, 'I':
0.07931034482758621, 'K': 0.03793103448275862, 'L':
0.1103448275862069, 'M': 0.013793103448275862, 'N':
0.03103448275862069, 'P': 0.041379310344827586, 'Q':
0.03793103448275862, 'R': 0.02413793103448276, 'S':
0.08275862068965517, 'T': 0.08620689655172414, 'V':
0.08620689655172414, 'W': 0.020689655172413793, 'Y':
0.05862068965517241}, {'A': 0.07228915662650602, 'C':
0.03614457831325301, 'D': 0.024096385542168676, 'E':
0.04819277108433735, 'F': 0.060240963855421686, 'G':
0.012048192771084338, 'H': 0.0, 'I': 0.03614457831325301, 'K':
0.024096385542168676, 'L': 0.18072289156626506, 'M':
0.012048192771084338, 'N': 0.060240963855421686, 'P':
0.024096385542168676, 'Q': 0.012048192771084338, 'R':
0.03614457831325301, 'S': 0.0963855421686747, 'T':
0.04819277108433735, 'V': 0.1566265060240964, 'W': 0.0, 'Y':
0.060240963855421686}, {'A': 0.015873015873015872, 'C': 0.0, 'D':
0.06349206349206349, 'E': 0.07936507936507936, 'F':
0.047619047619047616, 'G': 0.0, 'H': 0.015873015873015872, 'I':

```
0.15873015873015872, 'K': 0.06349206349206349, 'L':
0.12698412698412698, 'M': 0.047619047619047616, 'N':
0.06349206349206349, 'P': 0.015873015873015872, 'Q':
0.07936507936507936, 'R': 0.015873015873015872, 'S':
0.06349206349206349, 'T': 0.047619047619047616, 'V':
0.047619047619047616, 'W': 0.015873015873015872, 'Y':
0.031746031746031744}, {'A': 0.07317073170731707, 'C':
0.04878048780487805, 'D': 0.016260162601626018, 'E':
0.06504065040650407, 'F': 0.08130081300813008, 'G':
0.032520325203252036, 'H': 0.024390243902439025, 'I':
0.06504065040650407, 'K': 0.056910569105691054, 'L':
0.12195121951219512, 'M': 0.008130081300813009, 'N':
0.024390243902439025, 'P': 0.04878048780487805, 'Q':
0.040650406504065040, 'R': 0.040650406504065040, 'S':
0.056910569105691054, 'T': 0.08943089430894309, 'V':
0.06504065040650407, 'W': 0.0, 'Y': 0.040650406504065040}]
Результат роботи коду з розрахунку молекулярної ваги:
Molecular weight =

[5313.909099999998, 5088.881, 5483.274599999999,
5044.614799999999, 305149.810100002, 4689.4609999999975,
32785.399100000046, 9284.872599999993, 7528.714999999998,
13959.220099999982, 4796.7425, 4838.4766, 4832.543699999999]
Результат роботи коду з розрахунку ароматичності:
 Aromaticity =
 [0.13043478260869565, 0.175, 0.2391304347826087,
0.18604651162790697, 0.11662347278785636, 0.09523809523809523,
0.1275862068965517, 0.12048192771084337, 0.09523809523809523,
0.12195121951219512, 0.12195121951219513, 0.06976744186046512,
0.06976744186046512]
Результат роботи коду з розрахунку гнучкості білків:
Flexibility =  [1.024761904761905, 0.960797619047619,
0.9851547619047619, 1.0093214285714285, 0.9928333333333335,
1.0282380952380954, 1.0132023809523811, 1.018392857142857,
1.0311547619047619, 1.029059523809524, 1.0048809523809523,
1.0074166666666666, 0.9627619047619047, 0.9858928571428571,
0.9519880952380952, 0.9654166666666667, 0.9850952380952381,
0.9874999999999998, 0.9975595238095238, 0.9956428571428572,
1.0078928571428571, 0.9997619047619049, 0.9988095238095237,
0.9788809523809524, 0.9518809523809524, 0.9632380952380953,
0.9474642857142858, 0.9758095238095239, 0.9518333333333334,
0.9903333333333334, 0.9936785714285712, 0.995095238095238,
0.9979761904761905, 0.9770714285714287, 0.989642857142857,
1.0060833333333332, 0.9990595238095239]
Результат роботи коду з розрахунку ізоелектричної точки (pI)
білків:
Isoelectric point =
[11.52464237213135, 10.672499275207517, 6.421862602233887,
9.046284294128416, 6.899025917053222, 7.792374229431152,
5.672781562805175, 5.056362724304199, 4.677701377868652,
8.222894096374514, 9.504461097717286, 8.738576698303223,
8.65766887664795]
```

```
Результат роботи коду з розрахунку фракції вторинної структури
білків:
Secondary structure fraction =
[(0.2173913043478261, 0.2826086956521739, 0.15217391304347827),
(0.375, 0.15000000000000002, 0.125), (0.41304347826086957,
0.2391304347826087, 0.15217391304347827), (0.4186046511627907,
0.3488372093023256, 0.09302325581395349), (0.33543132173269163,
0.21177341725286933, 0.21325435024065165), (0.3333333333333333,
0.2857142857142857, 0.23809523809523808), (0.40344827586206894,
0.20344827586206898, 0.21724137931034482), (0.49397590361445787,
0.1927710843373494, 0.3132530120481928), (0.42857142857142855,
0.14285714285714285, 0.2698412698412698), (0.3739837398373984,
0.1626016260162602, 0.2682926829268293), (0.5365853658536586,
0.21951219512195125, 0.2682926829268293), (0.3023255813953488,
0.20930232558139536, 0.16279069767441862), (0.23255813953488372,
0.23255813953488372, 0.2558139534883721)]
Результат роботи коду з розрахунку індексу нестабільності білків:
 [46.2391304347826, 63.9375, 34.10434782608696, 33.93255813953488,
30.69366901147755, 73.21190476190478, 35.46758620689655,
33.98084337349397, 33.547619047619044, 46.80487804878048,
29.063414634146344, 44.246511627906976, 73.85348837209303]
```

Завдання до розділу «Пакет Biopython»

1. Завантажте з NCBI нуклеотидну послідовність білка mamE магнітотаксисної бактерії *Magnetospirillum gryphiswaldense* MSR-1 та збережіть файл з інформацією про цю послідовність у .fasta форматі. Визначить кількість нуклеотидів в цій послідовності.
2. Здійсніть трансляцію нуклеотидної послідовності білка mamE магнітотаксисної бактерії *Magnetospirillum gryphiswaldense* MSR-1 та розрахуйте довжину трансльованої послідовності.
3. Завантажте з NCBI нуклеотидну послідовність білка mamE магнітотаксисної бактерії *Magnetospirillum gryphiswaldense* MSR-1 та збережіть файл з інформацією про цю послідовність у .fasta форматі. Знайдіть в нуклеотидній послідовності білка mamE магнітотаксисної бактерії *Magnetospirillum gryphiswaldense* MSR-1 кількість кожного нуклеотиду.
4. Здійсніть трансляцію нуклеотидної послідовності білка mamE магнітотаксисної бактерії *Magnetospirillum gryphiswaldense* MSR-1 з урахуванням стандартного генетичного коду. Знайдіть в отриманій амінокислотній послідовності білка mamE магнітотаксисної бактерії *Magnetospirillum gryphiswaldense* MSR-1 кількість кожної амінокислоти.
5. Знайдіть в послідовності білка mamE магнітотаксисної бактерії *Magnetospirillum gryphiswaldense* MSR-1 загальну кількість амінокислот і кількість підпослідовностей GT.

Список рекомендованої літератури

1. Sebesta R. Concepts of Programming Languages / Robert Sebesta. – New Jersey : Pearson Education, 2012. – 816 с.
2. Chowdhary K. R. On the Evolution of Programming Languages [Електронний ресурс] / K. R. Chowdhary // arXiv. – 2020. – С. 1–6. – Режим доступу: https://doi.org/10.48550/arXiv.2007.02699 (дата звернення: 05.09.2023). – Назва з екрана.
3. Salus P. Handbook of Programming Languages, Volume II Imperative Programming Languages / Peter Salus. – Indianapolis : Macmillan Technical Publishing, 1998. – 473 с.

4. Kizior R. J. Does COBOL Have a Future? [Електронний ресурс] / R. J. Kizior, D. Carr, P. Halpern. – Режим доступу: https://web.archive.org/web/20160817115437/http://proc.isecon.org/2000/126/ISECON.2000.Kizior.pdf (дата звернення: 06.09.2023). – Назва з екрана.
5. Salus P. Handbook of Programming Languages, Volume I: Object-oriented programming languages / Peter Salus. – Indianapolis : Macmillan Technical Publishing, 1998. – 985 с.
6. Characterization and Identification of Programming Languages [Електронний ресурс] / Júlio Alves [та ін.] // OpenAccess Series in Informatics. – 2023. – С. 1–13. – Режим доступу: https://doi.org/10.4230/OASIcs.SLATE.2023.13 (дата звернення: 06.09.2023). – Назва з екрана.
7. Salus P. Handbook of Programming Languages, Volume III Little Languages and Tools / Peter Salus. – Indianapolis : Macmillan Technical Publishing, 1998. – 727 с.
8. Cravcenco D. From java to python: navigating the world of programming languages / D. Cravcenco, D. Bujilov, A. Ciornii // Conferinţa Tehnico-Conferinţa Tehnico-Ştiinţifică a Studenţilor, Masteranzilor și Doctoranzilor, Universitatea Tehnică a Moldovei, 5–7 квіт. 2023 р. – [Б. м.], 2023. – С. 1–5.
9. Python Software Foundation. Python [Електронний ресурс] / Python Software Foundation. – Режим доступу: https://www.python.org/ (дата звернення: 06.09.2023). – Назва з екрана.
10. Salus P. Handbook of Programming Languages, Volume IV Functional and Logic Programming Languages / Peter Salus. – Indianapolis : Macmillan Technical Publishing, 1998. – 281 с.
11. Яковенко, А. В. Основи програмування. Python. Частина 1 [Електронний ресурс] : підручник для студентів які навчаються за спеціальністю 122 «Комп'ютерні науки» спеціалізацією «Інформаційні технології в біології та медицині» / А. В. Яковенко ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1файл: 1,71 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 195 с. – Назва з екрана.
12. Python Software Foundation. Python 3.11.8 documentation [Електронний ресурс] / Python Software Foundation. – Режим доступу: https://docs.python.org/3.11/ (дата звернення: 06.09.2023). – Назва з екрана.
13. Костюченко А. О. Основи програмування мовою Python : навч. посіб. / А. О. Костюченко. – Чернігів : ФОП Баликіна С.М., 2020. – 180 с.
14. Python Software Foundation. Вбудовані константи [Електронний ресурс] / Python Software Foundation. – Режим доступу: https://docs.python.org/uk/3/library/constants.html#built-in-consts (дата звернення: 10.09.2023). – Назва з екрана.
15. Python Software Foundation. Вбудовані функції [Електронний ресурс] / Python Software Foundation. – Режим доступу: https://docs.python.org/uk/dev/library/functions.html (дата звернення: 11.09.2023). – Назва з екрана.
16. Громова В. Вбудовані функції [Електронний ресурс] / В. Громова. – Режим доступу: https://kpi-fpm-pma.github.io/python-labs-book/content/intro/functions.html (дата звернення: 11.09.2023). – Назва з екрана.
17. Python Software Foundation. Built-in Functions [Електронний ресурс] / Python Software Foundation. – Режим доступу: https://docs.python.org/3.9/library/functions.html#help (дата звернення: 11.09.2023). – Назва з екрана.
18. Python Software Foundation. Ітератори [Електронний ресурс] / Python Software Foundation. – Режим доступу: https://docs.python.org/uk/3.12/howto/functional.html#iterators (дата звернення: 15.09.2023). – Назва з екрана.
19. Python Software Foundation. Вбудовані типи [Електронний ресурс] / Python Software Foundation. – Режим доступу: https://docs.python.org/uk/3.12/library/stdtypes.html#iterator.__next__ (дата звернення: 15.09.2023). – Назва з екрана.
20. Копей В. Б. Мова програмування Python для інженерів і науковців : навч. посіб. / В. Б. Копей. – Івано-Франківськ : ІФНТУНГ, 2019. – 272 с.

21. Gowrishankar S. Introduction to Python Programming: Introduction to Python Programming / S. Gowrishankar, A. Veena. – Boca, Raton, London, New York : CRC Press Taylor & Francis Group, 2018. – 465 с.
22. Замуруєва О. В. Об'єктно-орієнтоване програмування в Python : курс лекцій / О. В. Замуруєва, А. С. Кримусь, Н. В. Ольхова. – Луцьк : Вежа-Друк, 2018. – 64 с.
23. Johansson R. Numerical python: Scientific computing and data science applications with numpy, SciPy and matplotlib, Second edition: Numerical Python Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib, Second Edition. / R. Johansson. – Chiba : Apress, 2019. – 709 с.
24. NumPy team. NumPy [Електронний ресурс] / NumPy team. – Режим доступу: https://numpy.org/ (дата звернення: 18.10.2023). – Назва з екрана.
25. Blanco-Silva F. J. Mastering SciPy / F. J. Blanco-Silva. – [Б. м.] : PRACT Publishing, 2015. – 483 с.
26. Yim A. Matplotlib for Python Developers / A. Yim, C. Chung, A. Yu. – Birmingham : Packt Publishing, 2018. – 297 с.
27. Ayars E. Computational Physics With Python / E. Ayars. – Chico : California State University, 2013. – 194 с.
28. The Matplotlib development team. Matplotlib: Visualization with Python [Електронний ресурс] / The Matplotlib development team. – Режим доступу: https://matplotlib.org/ (дата звернення: 21.10.2023). – Назва з екрана.
29. McGreggor D. M. Mastering matplotlib / D. M. McGreggor. – Birmingham, Mumbai : Packt Publishing, 2015. – 292 с.
30. Poladi S. Mastering matplotlib / S. Poladi. – Birmingham, Mumbai : Packt Publishing, 2018. – 676 с.
31. Bhasin H. Python Basics / H. Bhasin. – Dulles, Virginia Boston, Massachusetts, New Delhi : MERCURY LEARNING AND INFORMATION, 2019. – 566 с.
32. W3Schools. HTML Color Names [Електронний ресурс] / W3Schools. – Режим доступу: https://www.w3schools.com/colors/colors_names.asp (дата звернення: 22.10.2023). – Назва з екрана.
33. W3Schools. Colors HEX [Електронний ресурс] / W3Schools. – Режим доступу: https://www.w3schools.com/colors/colors_hexadecimal.asp (дата звернення: 22.10.2023). – Назва з екрана.
34. The Matplotlib development team. Choosing Colormaps [Електронний ресурс] / The Matplotlib development team. – Режим доступу: https://matplotlib.org/stable/users/explain/colors/colormaps.html (дата звернення: 22.10.2023). – Назва з екрана.
35. Scipy Lecture Notes / Gael Varoquaux [та ін.]. – Zenodo : HAL, 2015. – 368 с.
36. Scikit-learn developers. User Guide [Електронний ресурс] / Scikit-learn developers. – Режим доступу: https://scikit-learn.org/stable/user_guide.html (дата звернення: 29.10.2023). – Назва з екрана.
37. Boisberranger J. d. Scikit-learn: Machine Learning in Python [Електронний ресурс] / J. du Boisberranger, J. Van den Bossche, L. Estève. – Режим доступу: https://scikit-learn.org/stable/ (дата звернення: 30.10.2023). – Назва з екрана.
38. Morgan P. Data analysis from scratch with Python. Step By Step Guide / P. Morgan. – [Б. м.] : Journal of Chemical Information and Modeling, 2016. – 104 с.
39. Gradient Magnetic Field Accelerates Division of E. coli Nissle 1917 [Електронний ресурс] / Svitlana Gorobets [та ін.] // Cells. – 2023. – Т. 12, № 2. – С. 315. – Режим доступу: https://doi.org/10.3390/cells12020315 (дата звернення: 30.10.2023). – Назва з екрана.
40. Modeling of the Bacterial Growth Curve [Електронний ресурс] / M. H. Zwietering [та ін.] // Applied and Environmental Microbiology. – 1990. – Т. 56, № 6. – С. 1875–1881. – Режим доступу: https://doi.org/10.1128/aem.56.6.1875-1881.1990 (дата звернення: 30.10.2023). – Назва з екрана.

41. Doxygen. OpenCV: Open Source Computer Vision [Електронний ресурс] / Doxygen. – Режим доступу: https://docs.opencv.org/4.5.2/d6/d00/tutorial_py_root.html (дата звернення: 01.11.2023). – Назва з екрана.
42. Kaehler A. Learning OpenCV: Computer Vision with the OpenCV Library / Adrian Kaehler, Gary Bradski. – [Б. м.] : O'Reilly Media, Incorporated, 2008. – 577 с.
43. OpenCV Team. OpenCV [Електронний ресурс] / OpenCV Team. – Режим доступу: http://opencv.org (дата звернення: 30.10.2023). – Назва з екрана.
44. The Biopython Contributors. Biopython [Електронний ресурс] / The Biopython Contributors. – Режим доступу: http://www.biopython.org (дата звернення: 02.11.2023). – Назва з екрана.
45. Chang J. Biopython Tutorial and Cookbook [Електронний ресурс] / J. Chang, B. Chapman, I. Friedberg. – Режим доступу: https://biopython.org/DIST/docs/tutorial/Tutorial.pdf (дата звернення: 02.11.2023). – Назва з екрана.
46. NCBI. National Center for Biotechnology Information [Електронний ресурс] / NCBI. – Режим доступу: https://www.ncbi.nlm.nih.gov/ (дата звернення: 02.11.2023). – Назва з екрана.
47. NCBI. Entrez Programming Utilities Help - NCBI Bookshelf [Електронний ресурс] / NCBI. – Режим доступу: https://www.ncbi.nlm.nih.gov/books/NBK25501/ (дата звернення: 03.11.2023). – Назва з екрана.
48. A SARS-CoV-2 protein interaction map reveals targets for drug repurposing [Електронний ресурс] / David E. Gordon [та ін.] // Nature. – 2020. – Т. 583, № 7816. – С. 459–468. – Режим доступу: https://doi.org/10.1038/s41586-020-2286-9 (дата звернення: 03.11.2023). – Назва з екрана.
49. Biopython. Analyzing protein sequences with the ProtParam module [Електронний ресурс] / Biopython. – Режим доступу: https://biopython.org/wiki/ProtParam (дата звернення: 03.11.2023). – Назва з екрана.
50. Lobry J. R. Hydrophobicity, expressivity and aromaticity are the major trends of amino-acid usage in 999 Escherichia coli chromosome-encoded genes [Електронний ресурс] / J. R. Lobry, C. Gautier // Nucleic Acids Research. – 1994. – Т. 22, № 15. – С. 3174–3180. – Режим доступу: https://doi.org/10.1093/nar/22.15.3174 (дата звернення: 04.11.2023). – Назва з екрана.
51. Vihinen M. Accuracy of protein flexibility predictions [Електронний ресурс] / Mauno Vihinen, Esa Torkkila, Pentti Riikonen // Proteins: Structure, Function, and Genetics. – 1994. – Т. 19, № 2. – С. 141–149. – Режим доступу: https://doi.org/10.1002/prot.340190207 (дата звернення: 04.11.2023). – Назва з екрана.
52. Intrinsically disordered protein [Електронний ресурс] / A. Keith Dunker [та ін.] // Journal of Molecular Graphics and Modelling. – 2001. – Т. 19, № 1. – С. 26–59. – Режим доступу: https://doi.org/10.1016/s1093-3263(00)00138-8 (дата звернення: 04.11.2023). – Назва з екрана.
53. Protein Disorder and the Evolution of Molecular Recognition / A. Keith Dunker [та ін.] // Pac. Symp. Biocomput. – С. 473–484.
54. Improved amino acid flexibility parameters [Електронний ресурс] / David K. Smith [та ін.] // Protein Science. – 2003. – Т. 12, № 5. – С. 1060–1072. – Режим доступу: https://doi.org/10.1110/ps.0236203 (дата звернення: 05.11.2023). – Назва з екрана.
55. Bhaskaran R. Dynamics of amino acid residues in globular proteins [Електронний ресурс] / R. Bhaskaran, P. K. Ponnuswamy // International Journal of Peptide and Protein Research. – 2009. – Т. 24, № 2. – С. 180–191. – Режим доступу: https://doi.org/10.1111/j.1399-3011.1984.tb00944.x (дата звернення: 05.11.2023). – Назва з екрана.

56. Guruprasad K. Correlation between stability of a protein and its dipeptide composition: a novel approach for predicting in vivo stability of a protein from its primary sequence [Електронний ресурс] / Kunchur Guruprasad, B. V. Bhasker Reddy, Madhusudan W. Pandit // "Protein Engineering, Design and Selection". – 1990. – Т. 4, № 2. – С. 155–161. – Режим доступу: https://doi.org/10.1093/protein/4.2.155 (дата звернення: 05.11.2023). – Назва з екрана.
57. Rogers S. Amino acid sequences common to rapidly degraded proteins: the PEST hypothesis [Електронний ресурс] / S. Rogers, R. Wells, M. Rechsteiner // Science. – 1986. – Т. 234, № 4774. – С. 364–368. – Режим доступу: https://doi.org/10.1126/science.2876518 (дата звернення: 05.11.2023). – Назва з екрана.