Becoming an eLearning Hacker

An Intro to APIs, Web Scraping, and Learning Analytics

Royce Kimmons

Table of Contents

EdTech Books

Royce Kimmons

Brigham Young University

Royce Kimmons is an Associate Professor of Instructional Psychology and Technology at Brigham Young University where he studies digital participation divides specifically in the realms of social media, open education, and classroom technology use. He is also the founder of EdTechBooks.org. More information about his work may be found at http://roycekimmons.com, and you may also dialogue with him on Twitter @roycekimmons.

👍 Like this? Endorse it and let others know.

Endorse

Glossary

> 🔍 Find something...

Attribute (HTML)

Information about an HTML element that is included in the start tag as a key-value pair.

Content Management System (CMS)

A web-based application that organizes and provides content to users for viewing, editing, etc., typically consisting of a front-end that uses HTML/CSS/JavaScript and a backend that stores data in a database.

Document Object Model (DOM)

The logical tree of a web page that consists of organized HTML elements that may be listed or nested within one another.

Element (HTML)

A block of HTML that begins with a start tag, ends with an end tag, and encapsulates either text content or additional HTML elements. Denoted by angle brackets.

Float

A numeric value that includes decimal values, in contrast to integers (e.g., 3.14, 1.0, 0.8888).

Function

A reusable block of code that performs a specific task.

Integer

A numeric value that does not include decimal values, in contrast to floats (e.g., 0, 1, 27, 43).

Integrated Development Environment (IDE)

A software application that provides tools for software development.

Learning Management System (LMS)

A CMS with features specifically designed for learning management applications.

String

A series of text, such as a word, phrase, or sentence. A string can contain textual characters, special characters, and/or numbers.

Projects

This book was designed to be used in conjunction with formal learning activities focused on eLearning development. In this project-based approach, learners are expected to complete the following projects as tasks that give meaningful context to the instructional content of the book.

Scrape Internet Data
Create a Helpful JavaScript Function
Develop a Micro eLearning Module

Scrape Internet Data

Data   Internet   Web Scraping

Task

Identify a research question you'd like to answer by collecting a large amount of data from the Internet. Then, generate a Google Sheet or Microsoft Excel spreadsheet of a set of Internet data from one or more sites or from an API. Once you have collected the data, use the spreadsheet to recode and manipulate the data to answer your research question.

Requirements

To successfully complete this project, you must do the following:

- Adhere to relevant legal and ethical requirements for this project, including adhering to terms of use requirements, copyright laws, and anti-hacking laws.
- Use JavaScript and a web browser inspector or IDE to collect the data.
- Include at least 1,000 lines of initial data.
- Parse and structure the data appropriately to answer your question.
- Provide additional analysis in the spreadsheet and an answer to your question.
- Disclose how (if at all) you used AI.

Example Research Questions

You may ask any question you like, as long as it is answerable using internet data. Some examples might include the following:

- What trends exist for research articles published on the topic of open education?
- What external domains does a website link to?
- What percentage of images on a school's website have an appropriate `alt` attribute?
- How does the sentence complexity in a Jane Austen novel compare to modern novels?
- What is the frequency of feminine and masculine pronoun use in classical literature?

Create a Helpful JavaScript Function

JavaScript

Background

eLearning professionals find themselves working in a variety of online systems, but JavaScript tends to be a common language across most tools that use HTML/CSS that can be used for streamlining workflows and creating shortcuts.

Task

Create a JavaScript **function** that can be useful for other professionals or researchers. It should abide by legal and ethical requirements and should have one of the following contexts:

- A Standard Website
- Canvas
- EdTech Books

Then, test your function on at least 10 different pages in your context to ensure that it behaves the way you anticipate. Debug and improve your function as needed.

Examples

The possibilities here are endless, but some useful categories of functions might include the following:

- **Find** something on the page (e.g., images, headings)
- **Analyze** something on the page (e.g., word count, readability, accessibility features)
- **Change** something on the page (e.g., resize images, strip unwanted content, capitalize terms, replace double spaces).

In any case, be sure that you are providing an appropriate output for your function. For instance, if you are finding content, then you would want the output to show as a table or a list, while if you are changing content, you might just want to notify your user that the change occurred.

You can also combine *find*, *analyze*, and *change* functions, such as by analyzing the word count of a page in Canvas and then appending the results to the page so that readers can see it.

Develop a Micro eLearning Module

eLearning   HTML   CSS

Background

eLearning scenarios are commonly organized into short modules to allow for micro-learning and easier remixing. Any given module tends to focus on a small handful of learning objectives that may be accomplished in a relatively short amount of time, such as a single sitting, class session, or week.

Task

In JSFiddle or another **IDE**, create a short eLearning module that teaches a single concept or objective with an anticipated completion time of 10 minutes. The module may use existing third-party libraries, such as Bootstrap, and existing content as long as you are allowed to use the content. The module should include the following:

- At least one learning objective or goal
- At least four headings, and at least two levels of headings (e.g., H1 and H2)
- At least five paragraphs of text
- At least three images, diagrams, figures, or lists
- At least one table
- At least one callout box, blockquote, or other visually offset piece of content
- At least one embedded media object, such as a YouTube video

The module should rely upon CSS (or library-based) formatting whenever possible and should avoid common poor practices as described here.

Challenges

Challenges are short activities to help you apply what you are learning. They are intended to be done in conjunction with a course and to be shared with an instructor and peers. For many challenges, there may be multiple solutions, so the provided solutions are simply examples of how the challenge could be completed.

Hack a Website (well, sort of...)

Get All Email Addresses from a Staff Directory

Analyze Pronoun Usage in Tolstoy's Novels

Use LMS System Classes in Your Custom Content

Determine the Flesch-Kincaid Grade Level of Content on Any Website

Get the Navigation Outline of a Google Doc as a Table

Create a Bootstrap Web Page

Format a Poem Using Bootstrap

Format Images into a Grid Using Bootstrap

Format a Document Using Bootstrap

Remove Unnecessary HTML

Create Some JavaScript Listener Buttons

Dynamically Fetch Content via a REST API and Include it in Your Module

Use an AI (via a REST API) to Generate a Summary of Your Module's Content

Hack a Website (well, sort of...)

Changing Website Content and Formatting with the Inspector

HTML CSS

Figure 1

Example of a Screenshot of a Website Manipulated with the Inspector
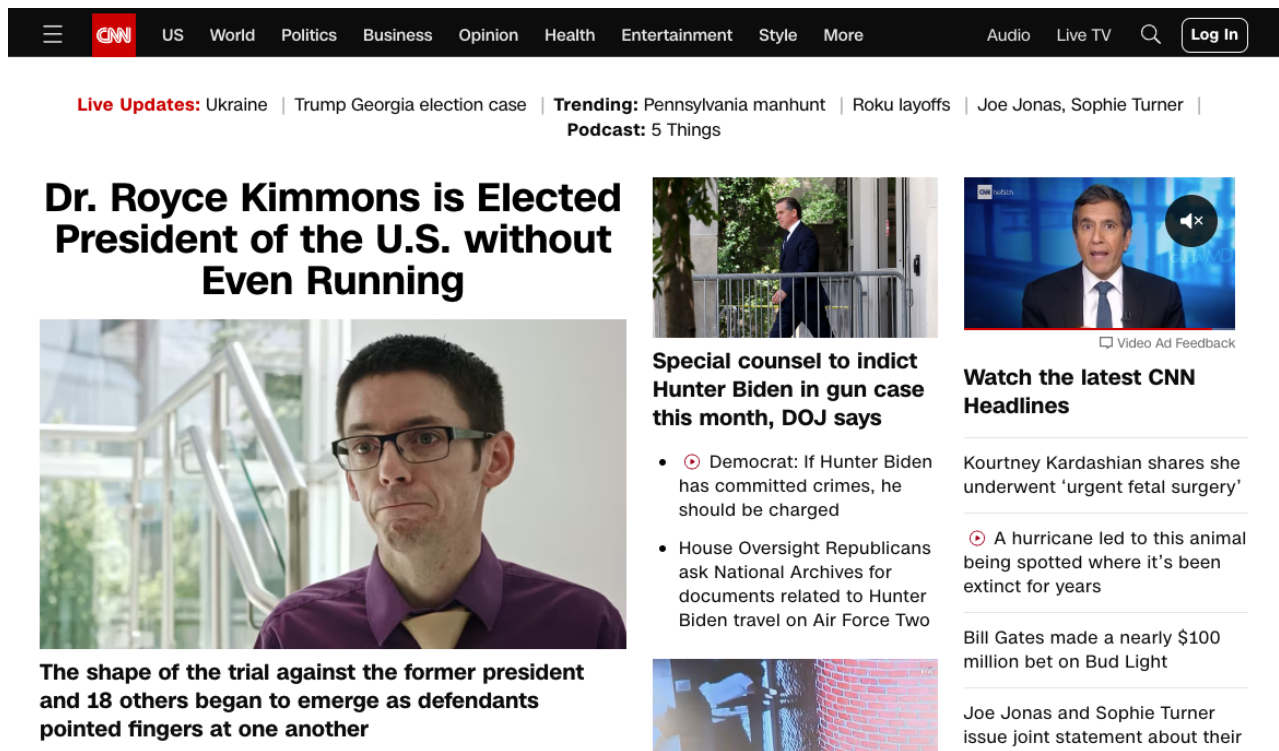
Background

Nowadays, people seem to think that a screenshot of a website is reliable evidence. Is this true? We need some examples of how websites can be manipulated in such a way that screenshots will provide misleading information.

Task

Using the Chrome Inspector, access 5 different websites and change their content or formatting in unexpected ways. Take screenshots of your changes, post them to a class discussion board, and briefly explain what you did. Try to be creative and do things that others have not thought of.

Get All Email Addresses from a Staff Directory

JavaScript   jQuery

Background

Your employer wants you to send an email to all faculty in the McKay School of Education, but you do not have a complete email list. You need everyone's email in the organization in a way that you can send them an email blast in Microsoft Outlook.

Task

Get all emails from a website as a comma-separated list.

Solution

There are many ways of doing this, but in looking at the webpage in the Inspector, notice that the cells with the email addresses all have the class "col-md-4". You also run `$()` in the Inspector Console and notice that jQuery is available. So, you might try running the following jQuery command in the Inspector Console to see if this at least gets the right data:

```
$('.col-md-4')
```

Though the results look accurate, there's also a lot of unnecessary spacing and returns in the data (e.g., "/n"). So, you can ask an AI the following prompt:

```
find the text of all '.col-md-4' elements and return them as an array with the contents trimmed of white space and returns
```

ChatGPT returns the following:

```
const colMd4Elements = document.querySelectorAll('.col-md-4');
const trimmedTextArray = [];
colMd4Elements.forEach(element => {
  const text = element.textContent.trim();
```

```
    trimmedTextArray.push(text);
});
console.log(trimmedTextArray);
```

Run this in the Inspector Console, and you will get the following results:

```
                                                                                    VM5534:14
(119) ['gekawika_allen@byu.edu', 'damon_bahr@byu.edu', 'dallinjb@byu.edu', 'david_barney@byu.edu', 'jeffbatt@gmail.com', 'dbaum@byu.edu', 'zack_beddoes@byu.
edu', 'brandan_beerli@byu.edu', 'david_boren@byu.edu', 'beth_borup@byu.edu', 'bryan.bowles@byu.edu', 'lorilynn_brandt@byu.edu', '', 'paul_caldarella@byu.ed
u', 'isaac_calvert@byu.edu', 'garrett.cardon@byu.edu', 'Nichole_Chaffee@byu.edu', 'peter.chan@byu.edu', 'cade_charlton@byu.edu', 'bchowen@byu.edu', 'lynnett
e_christensen@byu.edu', 'steven.christensen@byu.edu', 'sarah_clark@byu.edu', 'kaeli.cook@byu.edu', 'michael_cowan@byu.edu', 'elizabethcutrer@byu.edu', '',
'randy.davies@byu.edu', 'andrewdevey@byu.edu', 'dromey@byu.edu', 'lynnette_erickson@byu.edu', 'heidi.h.erickson@byu.edu', 'erika_feinauer@byu.edu', 'lane_fi
scher@byu.edu', 'adam_fisher@byu.edu', 'cally_flox@byu.edu', 'freire@byu.edu', 'terisa_gabrielsen@byu.edu', 'geo-jaja@byu.edu', 'agathag@byu.edu', 'tara_gou
lding@byu.edu', 'barry_graff@byu.edu', 'charles.graham@byu.edu', 'kendra_hall@byu.edu', 'pam_hallam@byu.edu', 'diane_hancock@byu.edu', 'blake_hansen@byu.ed
u', 'kristina_hansen@byu.edu', 'iptsec@byu.edu', 'Tyson_Harmon@byu.edu', 'kerstine_hart@byu.edu', 'hiltons@byu.edu', 'debrahogan@byu.edu', 'rebecca_hunter@b
yu.edu', 'james_huston@byu.edu', 'aaron_jackson@byu.edu', 'bryant_jensen@byu.edu', 'rkellems@byu.edu', 'lynne.kganetso@byu.edu', 'roycekimmons@byu.edu', 'he
ather_krieger@byu.edu', 'heather.leary@byu.edu', 'teresa_leavitt@byu.edu', 'michael.leonard@byu.edu', 'tracy_liu@byu.edu', 'darinl@byu.edu', 'corinne_maybe
ry@byu.edu', 'deidre_mccleery@byu.edu', 'jason@byu.edu', 'brandon.g.mcmillan@byu.edu', 'james_melville@byu.edu', 'nancy_miramontes@byu.edu', 'jared.morris@b
yu.edu', 'keri.moss@byu.edu', 'Melissa_Newberry@byu.edu', 'jeanna_nichols@byu.edu', 'stevan.lars.nielsen@byu.edu', 'beth_nisar@byu.edu', 'Shawn_Nissen@byu.e
du', 'rynixon@byu.edu', 'aaronolsen@byu.edu', 'johnpatten@byu.edu', 'todd.pennington@byu.edu', 'corinna_peterken@byu.edu', 'cecilia_pincock@byu.edu', 'miche
le_price@byu.edu', 'keven_prusak@byu.edu', 'peter_rich@byu.edu', 'michael_richardson@byu.edu', 'paul_ricks@byu.edu', 'lee_robinson@byu.edu', 'sarah.rollo@by
u.edu', 'alex_rosborough@byu.edu', 'christian_sabey@byu.edu', 'julieschow@byu.edu', 'jessie.scoville@byu.edu', 'nettina_smith@byu.edu', 'tbs@byu.edu', 'Barb
ara_smith@byu.edu', 'katie.stone@byu.edu', …]
```

This can then be copied into Outlook or elsewhere.

Analyze Pronoun Usage in Tolstoy's Novels

JavaScript

Background

Advocates for women's rights and representation in arts and literature regularly point out that women are often treated as subordinate to men or as taking supportive roles to men in books, movies, etc. Can we find evidence for this in classical literature? What about cases where women are the title characters of novels?

Task

Compare pronoun frequency and usage in two of Tolstoy's most well-known works, *War and Peace* and *Anna Karenina*, to see if masculine and feminine characters have similar treatment in each work. The full text of each work is available as HTML through [Project Gutenberg](#).

Solution

Most natural language processing tasks like this will require complex libraries, but we'll just ask an AI to provide some simple word frequency counts of the pronouns 'he', 'him', 'his', 'she', 'her', and 'hers'.

Here's an example of a prompt we could use:

```
using javascript in the inspector console, do a word frequency count on the contents of a web
page for the terms "he", "him", "his", "her", "hers", and "she" and provide results as a table
```

ChatGPT produced the following:

```
// Define the target words
    const targetWords = ['he', 'him', 'his', 'she', 'her', 'hers'];

    // Function to count the occurrences of target words in text
    function countWords(text) {
      const wordCounts = {};
      const words = text.toLowerCase().split(/\s+/); // Tokenize text by spaces
```
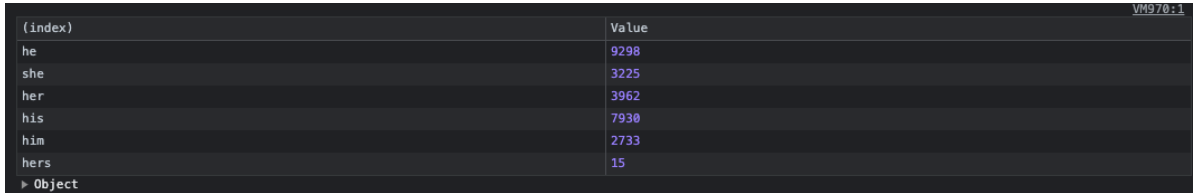
```
    for (const word of words) {
      if (targetWords.includes(word)) {
        wordCounts[word] = (wordCounts[word] || 0) + 1;
      }
    }

    return wordCounts;
  }

console.table(countWords(document.body.textContent))
```

For *War and Peace*, the results are as follows:

```
                                                         VM970:1
(index)                               Value
he                                    9298
she                                   3225
her                                   3962
his                                   7930
him                                   2733
hers                                  15
▶ Object
```
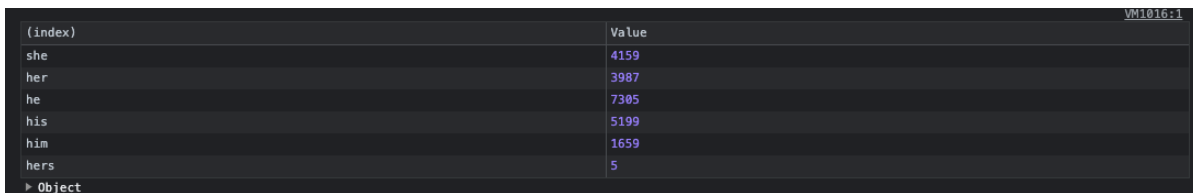
And here are the results for *Anna Karenina*:

```
                                                         VM1016:1
(index)                               Value
she                                   4159
her                                   3987
he                                    7305
his                                   5199
him                                   1659
hers                                  5
▶ Object
```

If we take these results and summarize them in a spreadsheet, we get the following:

	War and Peace		Anna Karenina	
	Frequency	**%**	**Frequency**	**%**
Masculine as Subject	9,298	74.25%	7,305	63.72%
Feminine as Subject	3,225	25.75%	4,159	36.28%
Masculine as Object	10,663	72.83%	6,858	63.21%
Feminine as Object	3,977	27.17%	3,992	36.79%

From this, we see that males seem to drive the narrative of each book, with a rate of 3-to-1 for *War and Peace* and 2-to-1 for *Anna Karenina*, despite the fact that the latter has a woman as the titular character.

Does this pattern continue in other classical works written by women, like for instance, Jane Austen or Emily Bronte?

Use LMS System Classes in Your Custom Content

CSS   LMS

Background

Like any website, **Learning Management Systems (LMS)** use CSS to format content, such as buttons, images, progress bars, etc. Because this CSS is already on the page, you can use it in your own content in LMS modules by using the correct classes.

For instance, if you inspect a button in Canvas, you will notice that it uses the "btn" class. If you use this same class in your own content, you can mimic the formatting.

Another example is icons. If you look at the HTML for a notification icon, you might see something like the following:

```
<i class="icon-announcement"></i>
```

This can then be used in your own content to use the same icon. Simply paste it into your inspector or into the HTML editor.

Task

Create some of your own content in a LMS and use the LMS's built-in CSS classes to format your content in ways not provided by the WYSIWYG editor. Include at least 5 class-based formatting elements from the LMS.

Determine the Flesch-Kincaid Grade Level of Content on Any Website

JavaScript

■ ■ ■

Background

The reading level of learning materials can impose unnecessary cognitive load and access barriers to learners. As a learning designer, you should ensure that any content you create or include in learning experiences is suitable for the reading level of your learners.

Task

Create an Inspector [Snippet](#) that you can run on any webpage to test the Flesch-Kincaid Grade Level of the webpage.

Solution

Here's an example of a prompt we could use:

```
using js in the inspector, calculate a web page's flesch-kincaid grade level score
```

In ChatGPT, this returned the following, which could be saved as a Snippet and run on any page:

```js
// Function to calculate the Flesch-Kincaid Grade Level score
function calculateFleschKincaid(text) {
  // Split the text into sentences
  const sentences = text.match(/[^.!?]+[.!?]+/g) || [];

  let totalWords = 0;
  let totalSyllables = 0;

  // Calculate the total number of words and syllables
  sentences.forEach((sentence) => {
```

```javascript
    const words = sentence.split(/\s+/);
    totalWords += words.length;

    words.forEach((word) => {
      totalSyllables += countSyllables(word);
    });
  });

  // Calculate the Flesch–Kincaid Grade Level score
  const averageWordsPerSentence = totalWords / sentences.length;
  const averageSyllablesPerWord = totalSyllables / totalWords;

  const score =
    0.39 * averageWordsPerSentence +
    11.8 * averageSyllablesPerWord –
    15.59;

  return score.toFixed(2);
}

// Function to count syllables in a word
function countSyllables(word) {
  word = word.toLowerCase().replace(/[^a–z]/g, '');
  if (word.length <= 3) return 1;
  word = word.replace(/(?:[^laeiouy]es|ed|[^laeiouy]e)$/g, '');
  word = word.replace(/^y/, '');
  return (word.match(/[aeiouy]{1,2}/g) || []).length;
}


// Get the text content of the web page
const textContent = document.body.textContent || '';

// Calculate the Flesch–Kincaid Grade Level score and log it to the console
const score = calculateFleschKincaid(textContent);
console.log(`Flesch–Kincaid Grade Level Score: ${score}`);
```

Get the Navigation Outline of a Google Doc as a Table

Background

Google Docs creates a navigation outline on the left sidetray that is useful for navigating a document. Your employer wants to see a copy of this outline in a Google Sheet so that they can more easily get a sense for the size and structure of the document.

Task

Using JavaScript in the Inspector Console, get the titles of a document's navigation outline items as a table that can be copy/pasted to a spreadsheet.

Solution

Here's an example function that could work:

```javascript
// Find all elements with the class "navigation-item"
const navigationItems = document.querySelectorAll('.navigation-item');

// Create an array to store the text content of the navigation items
const navigationText = [];

// Iterate through the elements and add their text content to the array
navigationItems.forEach((item) => {
    navigationText.push({
        'Navigation Item': item.textContent.trim()
    });
});

// Display the results in a table in the console
console.table(navigationText);
```

```
Console  Rendering  Issues  What's New

top ▼        Filter                                          Default levels ▼    6 Issues: 🚩 6

▶ ≡ 35 messages           < undefined
▶ ⊙ 3 user messages     > // Find all elements with the class "navigation-item"
▶ ⊗ 1 error               const navigationItems = document.querySelectorAll('.navigation-item');
▶ ⚠ 31 warnings
▶ ⓘ 3 info                 // Create an array to store the text content of the navigation items
  🐞 No verbose            const navigationText = [];

                           // Iterate through the elements and add their text content to the array
                           navigationItems.forEach((item) => {
                               navigationText.push({
                                   'Navigation Item': item.textContent.trim()
                               });
                           });

                           // Display the results in a table in the console
                           console.table(navigationText);
                                                                                    VM2883:15
```

(index)	Navigation Item
0	'EXECUTIVE SUMMARY'
1	'PROGRESS IN KEY AREAS SINCE THE LAST UNIT REVIEW'
2	'1. Actions Taken Based on Previous Review Recommendations'
3	'1.a. Making Advising Loads More Manageable and Transparent'
4	'1.b. Grooming New Departmental Leadership'
5	'2. Significant Changes in the Program, Faculty, or Operations'
6	'UNIT OVERVIEW'
7	'Organizational Chart'
8	'Mission and Strategic Objectives'
9	'Decision Making'
10	'Leadership and Administrative Succession Planning'
11	'Climate'

Create a Bootstrap Web Page

HTML   CSS   Bootstrap

Background

Often when working in an online system, you do not have access to edit the CSS itself, but you do have existing libraries within the system that you can draw upon to format your content. Bootstrap is one common library that exists on many web pages.

Task

In JSFiddle or another IDE, create a webpage that uses the Bootstrap library. Place some dummy content in your web page, and modify the HTML to show the following Bootstrap-formatted elements (without adding any CSS):

- A green success button
- A paragraph with a black background and white text
- A red alert message
- A yellow progress bar at 50% completion
- A rounded blockquote element with a shadow

Also be sure that you are using appropriate padding and margins throughout.

Solution

Format a Poem Using Bootstrap

Bootstrap

Background

Formatting textual and other content in HTML can seem tricky, but applying a few simple tags (e.g., `p`, `br`) and classes can make a huge difference!

Task

Format the following HTML into a nice-looking poem that properly separates stanzas. Also be sure to do the following:

- Make the title a heading.
- Italicize the subtitle and make it larger.
- Make the stanzas properly spaced (with spacing between every fourth line, but no spacing within the stanza).
- Apply reasonable margins and spacing to the body and to all elements on the page.
- Make the image a reasonable size.
- Do anything else you think would look good.

Solution

Format Images into a Grid Using Bootstrap

Bootstrap

Background

Organizing content horizontally on a webpage can be challenging, but Bootstrap gives us tools to do this more efficiently and in a way that is responsive to different screen sizes.

Task

Reformat the provided webpage into a structure that is more user-friendly. Do the following:

- Wrap all images into a single `div` element with class `row`.
- Make the first row of images show only three images and fill the whole screen.
- Make the next two rows show four images and fill the whole screen.
- Make any remaining rows of images show up six to a row.
- Add some padding.
- Make the first row of images change so that each image fills the screen when the screen width is smaller than `sm`.
- Do something else cool, too.

Solution

Format a Document Using Bootstrap

Bootstrap

Background

In eLearning development, you often have to reformat documents created in word processing or other applications for the web.

Task

Reformat this book chapter by doing the following:

- Adjust the notifications at the top of the page to only show on specific screen width breakpoints (hint: look up the `d-block` and `d-none` classes in Bootstrap)
- Make the first image fill the screen on all screen sizes.
- Make the second image flexible so that it fills the screen on small screen sizes but floats on medium and larger ones.
- Wrap the "Additional Readings" section in a callout box of some kind.
- Make the "Additional Readings" section float to the right on larger screen sizes.
- Make the table rows striped.
- Create a custom class called "reference" that applies a hanging indent to the reference item (hint: you'll need to use CSS for this).
- Make the YouTube video fill the screen and adapt to screen size (hint: look up the `ratio` class in Bootstrap).
- Adjust padding and margins to make it look better.
- Do anything else that you think would make it nicer.

Solution

Remove Unnecessary HTML

HTML

Background

You will often find yourself working with imported HTML that contains a lot of unnecessary information. This is a problem because it makes it harder to correct formatting mistakes and increases download sizes.

Task

In the following web page, remove all unnecessary HTML information (e.g., elements, attributes) to make the HTML use as few characters as possible without significantly changing its visual formatting. Check the console read out to see the number of characters on each run.

I reduced it from **5,098 characters** to **624 characters**, which is a reduction of **88%**. How about you?

Solution

Create Some JavaScript Listener Buttons

JavaScript   Bootstrap

Background

JavaScript Listeners can be very useful in eLearning development for allowing users to interact with a tool or its content. Listeners might trigger visible changes to the page or behaviors that the user notices, or they might also be used to store data about user behavior for analysis (as with click-level analytics).

Task

Create a Bootstrap HTML page with at least 4 buttons. Give each button a unique ID, and make each button do something different when it is clicked. For example, each button might show a different alert message. Or you can also have an AI help you write what you would like to occur by providing it a prompt ike "in JS, add a new class to an HTML element."

Do something creative and fun! Check out the solution Result tab for an example of a *Harry Potter* sorting hat.

Solution

Dynamically Fetch Content via a REST API and Include it in Your Module

JavaScript   JSON   REST API

Background

Often you may want to include content from external sources that might be updated or adjusted over time, or you might want to include a widget or another dynamic element in your content that you would like to stay up-to-date or that you would like to rebrand for your purposes. Rather than copying and pasting such elements into your system as static entities, you can dynamically fetch them when the page is loaded with a REST API call. You can also add listeners to only load contents at certain times, such as only loading a glossary term if a learner asks for it.

Task

In your module, dynamically fetch content from EdTech Books or another service using a REST API call and add the content to your module. A simple and useful case would be fetching the definition to a term when a learner clicks on it.

Generic Dictionary Example from RapidAPI

> Be sure to update the JavaScript to use your own API key. Otherwise, it will not work.

EdTech Books Example

This content is provided to you freely by EdTech Books.

Access it online or download it at
https://edtechbooks.org/elearning_hacker/dynamically_fetch_content_via_a_rest_api_and_include_it_in_your_modu

Use an AI (via a REST API) to Generate a Summary of Your Module's Content

AI   JavaScript   JSON   REST API

Background

There are all kinds of great AI tools out there that you can use in your work, and you can tap into many of them by using a REST API. In an eLearning module, it's often useful to provide a summary of your content, and this is something that many AIs are trained to do.

Task

Using a Large Language Model on Hugging Face (or another service), use a REST API call to summarize your module content. Then, provide the summary in a box at the top of your page as a sort of abstract. The BART Large CNN model is a good one to start with. Note that you will need to create an account with Hugging Face and get an access token before it will accept your data.

Example

Essential Tools

To complete the projects and activities in this book, you will benefit from the following free tools that will be referenced throughout.

Integrated Development Environments (IDEs)
Web Browser Inspectors
Generative Artificial Intelligences (AIs)

Integrated Development Environments (IDEs)

HTML   Web Development   CSS   JavaScript

An Integrated Development Environment (IDE) is a tool for creating and testing code. There are many free IDEs that are common, and you will need to use one to complete the projects. Some popular examples include the following:

- VisualStudio
- JSFiddle
- CodePen

Web Browser Inspectors

HTML   Web Development   CSS   JavaScript

Most modern web browsers have an inspector feature that allows you to see what's going on under-the-hood and to do some advanced things to web pages you have loaded, such as seeing their HTML source and running Javascript commands on loaded content. Inspectors are extremely helpful in eLearning work as they allow you to troubleshoot, test, design, and develop content and tools more easily.

Each browser is different, but this book will focus primarily on Chrome and the Chrome Inspector.

To access the Inspector, open a web page in your browser, right-click anywhere on the page, and choose Inspect.

Figure 1

Typical Inspector View
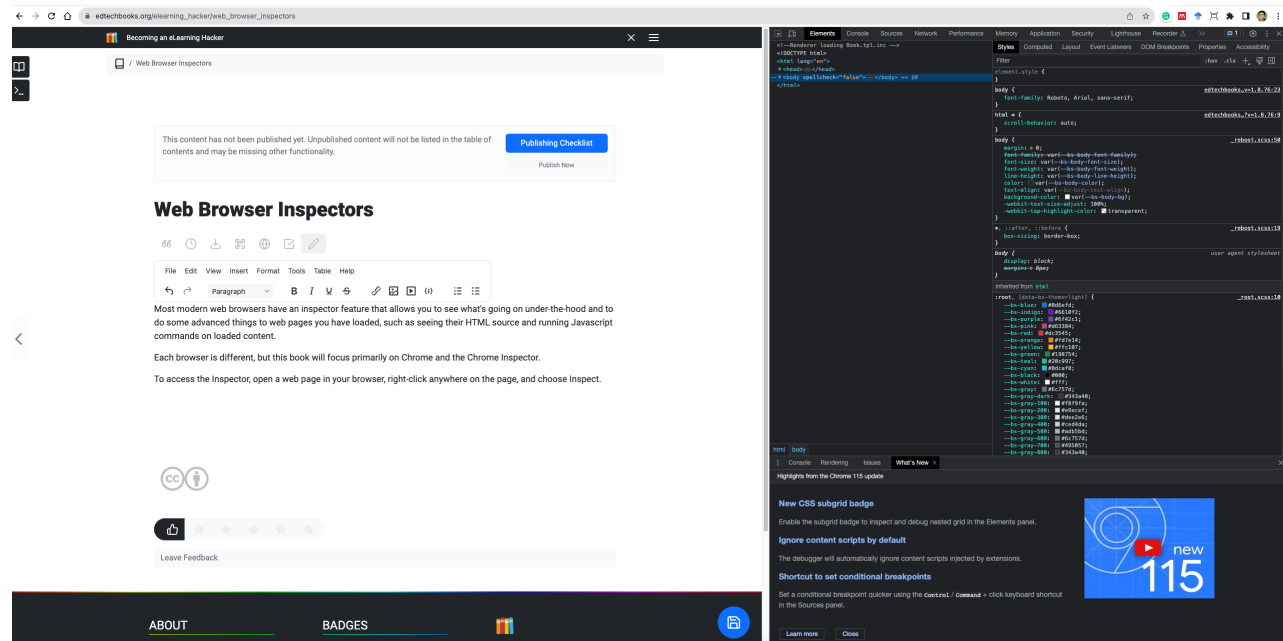
This will open a panel on either the side or bottom of the browser window. In this panel, you'll find many tabs and options. A few that are of most importance for us at this point include the following: HTML Source, Styles, Device View, and Console.

HTML Source

You can view the full HTML of the active page under the Elements tab. Expand or contract elements to see nested elements inside. You can also directly edit the HTML and see changes applied in the active page. You can also right-click on any element on the page and click Inspect, and it will jump to that element in the HTML Source View.

Figure 2

The HTML Source View and Editor
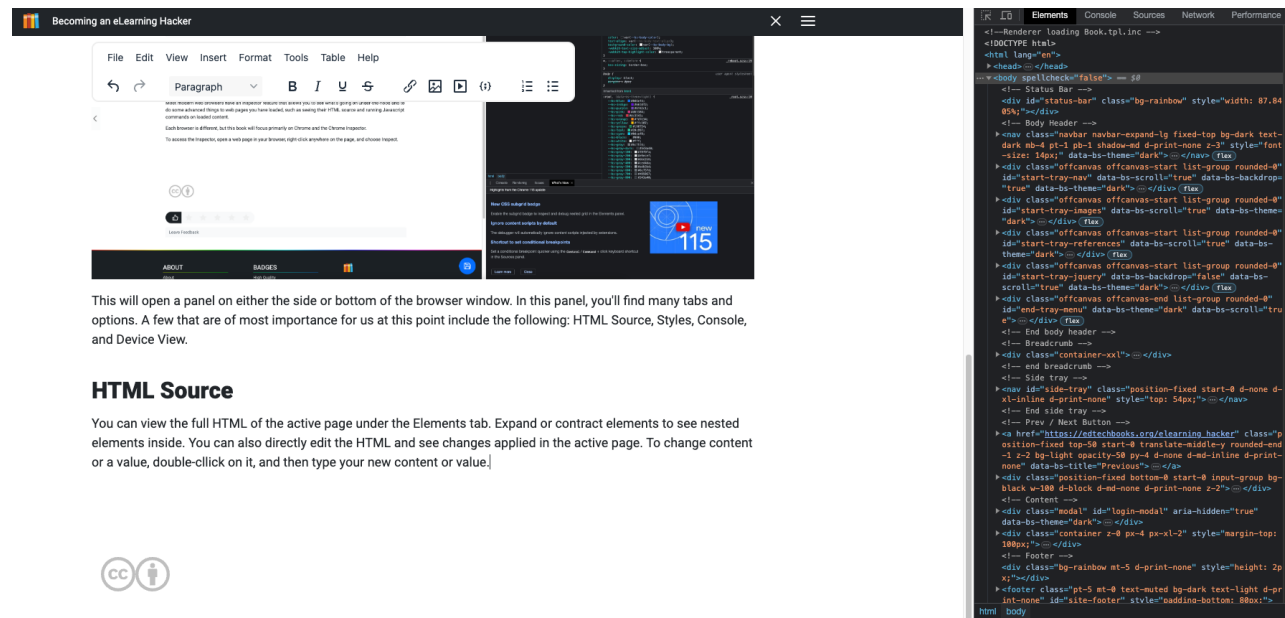
To change content or a value, double-click on it, and then type your new content or value. For instance, try double-clicking a `<p>` element and changing the text inside it.

The HTML Source Viewer and Editor is useful for seeing the structure of a page, finding problems, and extracting content.

Styles

The Styles panel is also found in the Elements tab but allows you to see and edit the active CSS styles of the element selected in the HTML Source Editor. To edit the HTML element's (and any similar elements') CSS, select the element in the HTML Source Editor and double-click in the `element.style{ }` area of the Styles panel. Then you can type any CSS you'd like.

Figure 3

The Styles Panel
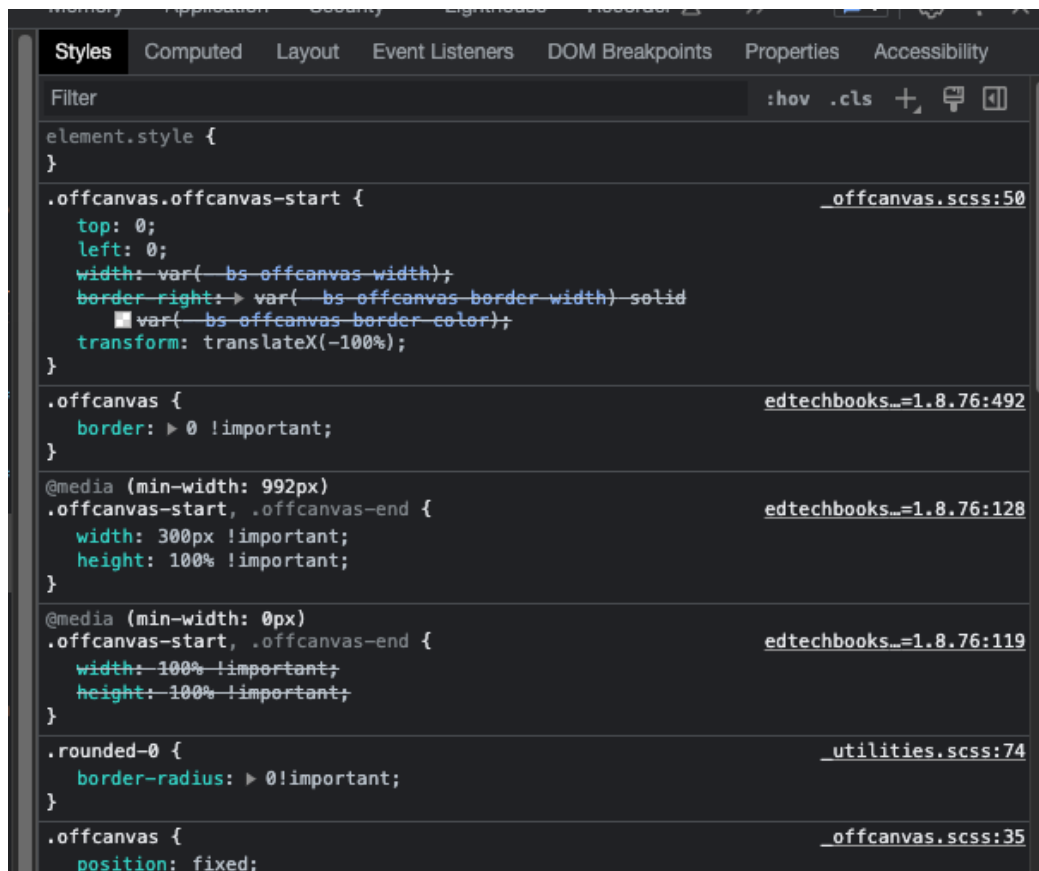
For instance, try selecting the `<body>` HTML element and in the `element.style { }` area, type `color: pink;` and hit enter. This should change the font color of all of your text on your page to pink (unless you have other styles applied to specific text elements).

In this panel, you can also see all of the CSS styles that are applied to each element. This is helpful for finding the source of formatting options and adjusting them across an entire site.

Device View

Since you are likely creating content for a variety of device types, it's helpful to be able to see how your content will be rendered on different screen sizes and device types. By clicking the `Toggle device toolbar` button, you will then be able to adjust the size of your viewing area to match your target user's device.

Try viewing the page with three or four different device settings and see how it effectively adapts (or fails to adapt) to the different views.
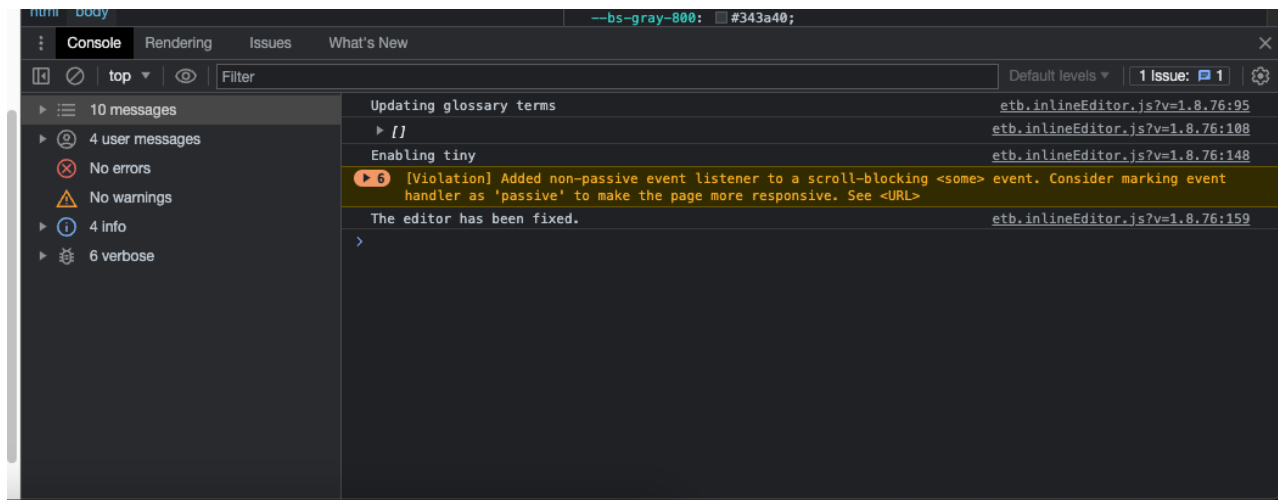
The Device View is essential for testing your product's accessibility and adaptability.

Console

The console allows you to run Javascript commands and to see errors and messages that are typically hidden from page viewers.

Figure 4

The Console

To run a command, click next to the carat and type your command. For instance, try typing `alert("Hello there!");` and hit enter. You should see a pop-up that says "Hello there!"

Also notice that the Console tracks your commands, so you can rerun or edit a previous command by simply pressing the up arrow key.

The Console is useful for developing Javascript functions for your applications, for testing website features, or for extracting content from a page.

Challenge: Find the Hidden Email Addresses

There are seven (7) hidden email addresses on this page. Can you find them using the inspector?

As a bonus, can you make them all visible? (Hint: The "d-none" class is what is making them invisible.)

Once you've found them, run the following JavaScript in the Inspector Console to see some magic happen:

```
$('.d-none').removeClass('d-none').css('font-size','30px').css('color','blue');
```

Snippets

In the Inspector, you can also store Snippets of JavaScript in the `Source > Snippets` panel. This is especially useful if you plan to run the same code over and over again on different webpages and allows you to build a library of solutions that you an easily access in the future.

Generative Artificial Intelligences (AIs)

Artificial Intelligence   HTML   Web Development   CSS   JavaScript   jQuery

Generative Artificial Intelligence (AI) has many uses for eLearning developers, but the main way I will be encouraging you to use it in this book and with these projects is to help you remember and generate HTML, CSS, JavaScript, and jQuery syntax.

These tools change and improve rapidly, and for the purposes of this book, you may use any AI that you have available to you. Some of the more popular ones at present include the following:

- ChatGPT
- GitHub Copilot
- CodeWhisperer

Forming Prompts for an AI

The quality of output you receive from an AI is directly influenced by the quality of prompt you provide. For our purposes, you generally should include four things in any prompt: the language or syntax, the target document or product, the desired action or format, and any parameters to operate within.

Here are a few examples of useful AI prompts to try:

> In JavaScript, select all links from a web page, and return their URLs as a comma-separated list.

> Create a boilerplate HTML document for an eLearning module that includes an external CSS, an external Javascript file, the jQuery library, and Bootstrap.

> Create a CSS definition for blockquotes that indents the text, makes it italicized, and changes the color to a dark grey.

> Create a jQuery function that detects when a link on the page is clicked and sends the event to an API for collection.

> In Bootstrap, make an accordion that expands and contracts when clicked. Then, send the state of the accordion as an event to the API.

Additionally, many AIs remember your previous prompts and can build upon them. So, if you used the third prompt above and then decided you wanted the color to be blue, you could just provide a prompt like "make the text blue

instead" to receive an updated version of the answer. If the output doesn't work the way you anticipate, you can also let the AI know about your unexpected behavior or any errors that you receive so that it can try to provide a better solution.

HTML

HTML   Web Development

HTML stands for Hypertext Markup Language, and it's made up of bracketed tags (or elements) that tell a web browser how to format and treat content.

Document Object Model (DOM)
Common HTML Elements
Common HTML Attributes
Cleaning HTML
Helpful HTML Snippets

Document Object Model (DOM)

HTML   JavaScript
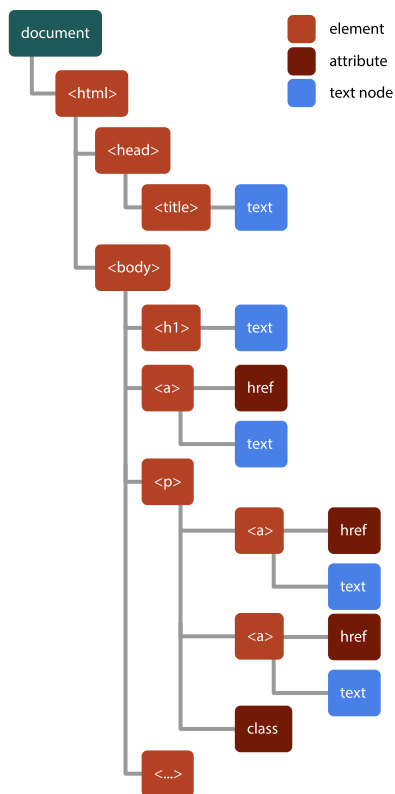
Webpages are structured as **Document Object Models** (DOMs), which just means that the content of any webpage is organized as nested **HTML elements**, as in Figure 1.

Figure 1

An Example of a Webpage Document Object Model (DOM)

Each element can have **attributes** that provide information about it and can encapsulate other elements or a text node, which has a **string**.

Figure 2

Structure of an HTML Element
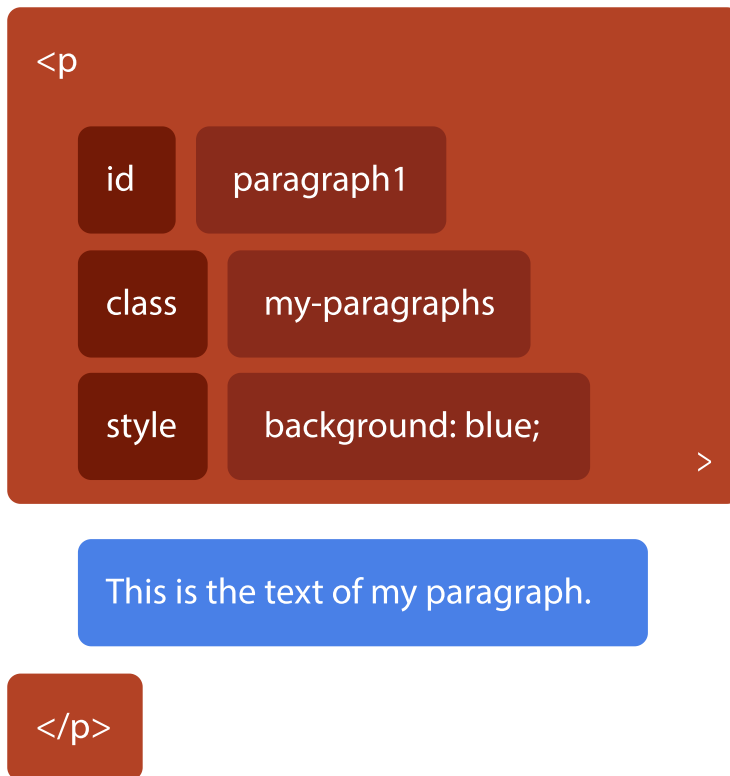
The DOM ensures that content on a web page is not just randomly represented but is displayed and accessed in a structured manner.

When working within a web-based editor as a content editor, such as a WYSIWYG editor in a **LMS** or **CMS**, you are typically only editing a very small part of the DOM, as in Figure 3.

Figure 3

The eLearning Content Editor's Editable Part of a DOM within a LMS or CMS

The caveat to this is that if you are able to run JavaScript on the page, then you can manipulate any part of the DOM as you would like. This is why many platforms restrict who can crea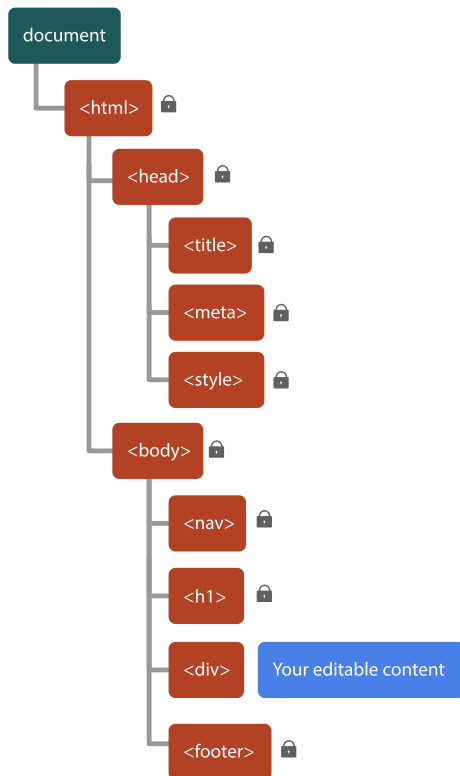te JavaScript that runs on the page, because if you can include some JavaScript in your editable area, then you can adjust content on the page outside of your editable area, such as the navigation, headings, styles, etc.

This content is provided to you freely by EdTech Books.

Access it online or download it at
https://edtechbooks.org/elearning_hacker/document_object_model.

Common HTML Elements

HTML   Web Development

There are currently about 150 HTML **element** types used on the web, but to be successful in eLearning work, there are only 20 or so that you really need to know. Here are the most important ones with brief descriptions.

Tag	Name	Description	Key Attributes
Document-Level Tags			
<head>	Header (Machine)	This header is invisible to users, includes metadata, loads necessary formatting for the page, and does other things behind the scenes.	
<title>	Title	This is used in the header to set the title of the page to be shown in the browser tab.	
<link>	Linked Files	This is used in the header to identify linked stylesheets that are shared between multiple pages.	src="[link to the file]"
<script>	JavaScript	This is used in the header or in the body to include JavaScript code or to include an external Javascript file.	src="[link to the file]"
<body>	Body	This is the main content area of a page.	
Content-Level Tags			
<article>	Article	Within the body, this is the main organizer for content. For instance, if you have a list of blog posts, each post would be a separate article.	
<h1> <h2> <h3> <h4> <h5> <h6>	Headings	Headings are used to designate titles and subtitles in a page, with H1 being the highest level (e.g., the title of the page) and H6 being the lowest (e.g., a section title nested deeply below other sections).	

Tag	Name	Description	Key Attributes
<a>	Link	A hyperlink to another page or section within a page.	href="[link to the target]"
<nav>	Navigation	A navigation menu or area.	
<p>	Paragraph	A paragraph of content.	
 	Break Return	A hard break in content. This should typically only be used if you need to separate content within a paragraph, such as with a poem. Notice also that the tag is self-closing, so no content should be inside it.	
<table>	Table	A content table (see Tables below).	
<form>	Form	A form for accepting input from users.	method="POST/GET" action="[URL]"
<input>	Input	Accepts user-submitted content for a form (e.g., a textfield).	value="[value]"
<button>	Button	An interactive element (typically used with JavaScript) to change the page in some way.	
<blockquote>	Blockquote	An indented paragraph of content.	
<em>	Emphasis or Italic	Italicizes content, making it slanted.	
<strong>	Strong or Bold	Bolds content, making it darker.	
<img>	Image	Embeds an image file.	src="[link to the file]" alt="[description]"
<audio>	Audio	Embeds an audio file.	src="[link to the file]" alt="[description]"
<video>	Video	Embeds a video file.	src="[link to the file]" alt="[description]"
	List	A list of content, either Unordered (using bullets) or Ordered (using numbers).	
<sup> <sub>	Superscript Subscript	These are used for exponents and scientific notation, such as $E=mc^2$ (superscript) and H_2O (subscript).	

Tables

Tables are great for organizing and visualizing content (especially data), but they use a number of specific tags in specific orders, and there are some basic rules you must follow to make sure that tables are accessible and rendered properly. In addition, tables **should not be used simply to visually layout content** (e.g., to place two images side by side). Instead, you should use CSS properties for strictly visual manipulation.

The tags you should know for tables include the following:

Tag	Name	Description
<table>	Table	This is the enclosing tag for the entire table.
<thead>	Header	This encloses all of the header content for the table, such as labels for columns.
<tbody>	Body	This encloses the body content of the table.
<tr>	Row	This encloses a row of cells.
<th>	Cell Heading	This is a heading cell.
<td>	Cell Content	This is a content cell.

Tables are laid out top to bottom, and then rows are laid out left to right. All tables should follow this basic structure:

```
<table>
  <caption>Here's the optional caption.</caption>
  <thead>
    <tr>
      <th></th>
      <th>Column A</th>
      <th>Column B</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>Row 1</th>
      <td>Content 1A</td>
      <td>Content 1B</td>
    </tr>
    <tr>
      <th>Row 2</th>
      <td>Content 2A</td>
      <td>Content 2B</td>
    </tr>
  </tbody>
</table>
```

This would render as follows:

	Column A	Column B
Row 1	Content 1A	Content 1B
Row 2	Content 2A	Content 2B

Here's the optional caption.

Visually, tables are laid out as follows:

Figure 1

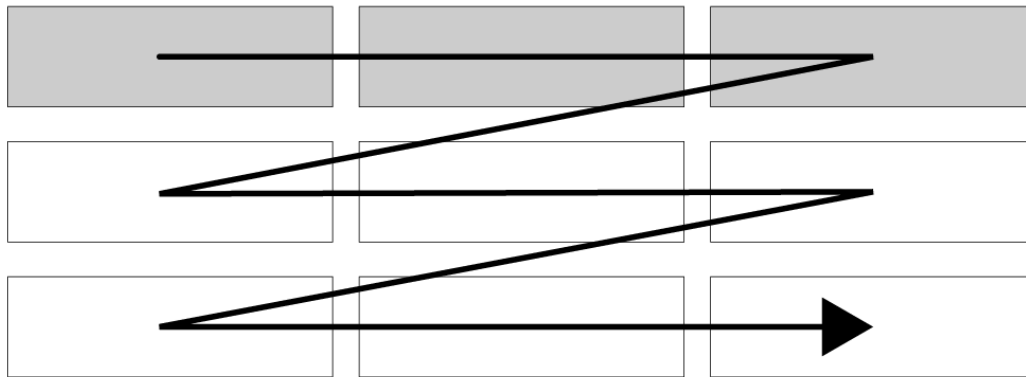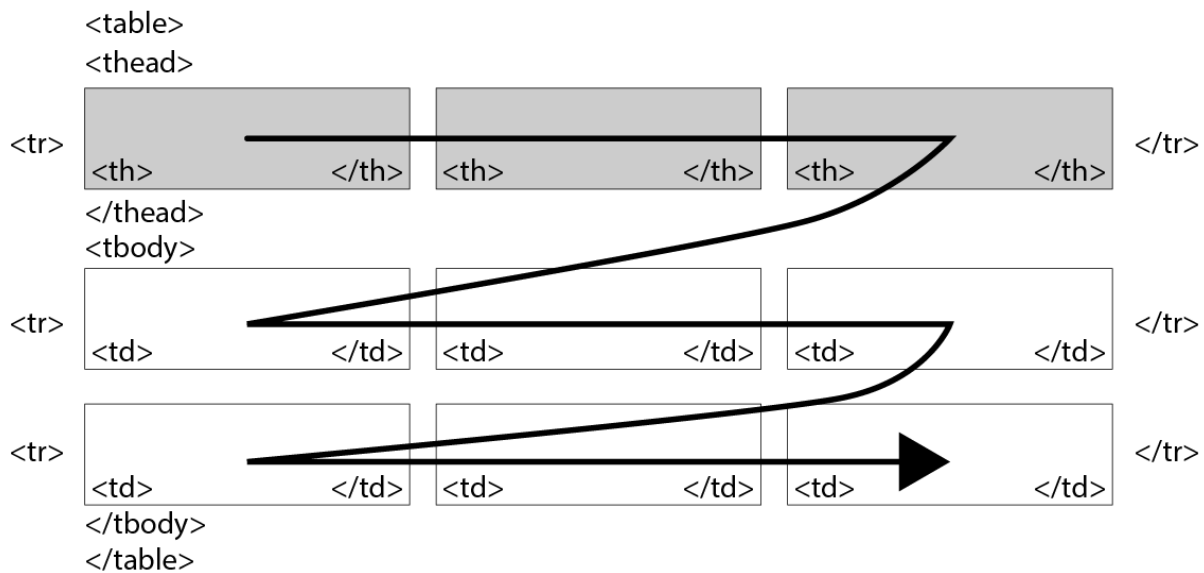*Visual layout of a table from top-to-bottom and left-to-right*

Figure 2

Visual layout of a table with tags

Here are a few things to notice:

1. The caption is included first but is actually rendered last. The position and style of the caption (first or last) depends on your CSS settings.
2. The first `<th>` element is intentionally left empty or blank but must be included so that the columns line up properly.
3. Instead of manually bolding or italicizing headers, we use the semantic `<th>` identifier instead of `<td>`. This communicates to indexers and screen readers that the heading applies to a column or row.

Lists

Lists all follow the same structure of defining the list with `<ul>` or `<ol>` tags and then including `<li>` tags for each list item within them. Here's an example:

```html
<ul>
  <li>Apple</li>
  <li>Banana</li>
  <li>Orange</li>
</ul>
```

This would render as follows:

- Apple
- Banana
- Orange

Changing the encapsulating `<ul>` to `<ol>` would make it numbered instead of bulleted, as follows:

1. Apple
2. Banana
3. Orange

To indent (or nest) lists, you simply include new lists within list items, as follows:

```html
<ol>
  <li>Fruits
    <ol>
      <li>Apple</li>
      <li>Banana</li>
    </ol>
  </li>
  <li>Vegetables
    <ol>
      <li>Carrot</li>
      <li>Broccoli</li>
    </ol>
  </li>
</ol>
```

This would render as follows:

1. Fruits
    1. Apple
    2. Banana
2. Vegetables
    1. Carrot
    2. Broccoli

Following this pattern, you can make infinitely nested lists as necessary. However, you should avoid including other tags within lists, such as `<p>`, `<img>`, etc. as they can quickly confuse lists and influence their structure and rendering.

Things to Avoid

Just as there are common HTML elements that everyone should know and use, there are also some that are typically misused. Here are some of the most important to be aware of with explanations.

Tag	Name	Description
<iframe>	IFrame	This embeds one page within another and is often used to embed external resources. It can be used properly, but it is often a security risk and poses problems for browsers because you cannot control the content within the frame.
<table>* * for layouts	Table	Tables are great for data, but sometimes people use them for laying out content (e.g., creating rows and columns of images). This can pose serious accessibility problems and should be avoided.
<span>	Span	Spans have no semantic meaning but basically allow you to apply formatting to a chunk of content. They can be useful for visually adjusting content, but screen readers and other machine applications have difficulty making sense of them. They are also often used frequently for no reason, which makes the HTML messy and bloated.
<div>	Division Layer	Similar to spans, divs have no semantic meaning, and though they can (and should) be used effectively in web development, most of the time eLearning and content developers don't need to use them.
<i>	Italic	The <em> tag is preferred to the <i> tag because it is more semantic. They appear the same to the reader.
<b>	Bold	The <strong> tag is preferred to the <b> because it is more semantic.They appear the same to the reader.
style=""	Style Attribute	Editing the style attribute of an HTML element to change its formatting is not always bad, but if you are changing style attributes across multiple elements in the same way (e.g., making the text larger in multiple paragraph elements), then you should instead rely upon CSS.

Common HTML Attributes

HTML   Web Development

Attributes are invisible key-value pairs that are included inside HTML elements (or tags) to assign information to the element, such as:

```
<p attribute-key="attribute-value">Example paragraph</p>
```

General Attributes

These general attributes are constantly used and are extremely useful.

id

Assign a unique identification **string** to an element. This is useful for referencing the element with CSS or JavaScript.

```
<p id="paragraph1">Example paragraph</p>
```

Once this paragraph is assigned the id "paragraph1" we can then write CSS either in our header or an external file to format it, as follows:

```
p#paragraph1 { color: blue; }
```

When referencing an element with an id in CSS, use the name of the element type (in this case, `p`) followed by the hashtag (#) symbol and the element's id. The result will be as follows:

> Example paragraph

class

Assign one or more class names to an element. This is useful for referencing the element as part of a group with CSS or JavaScript.

```
<p class="speaker1">Hello</p>
<p class="speaker2">Hi</p>
```

```
<p class="speaker1">How are you?</p>
<p class="speaker2">Fine. You?</p>
```

We can then assign CSS values as follows:

```
p.speaker1 { color: green; }
p.speaker2 { color: coral; }
```

When referencing an element with a class in CSS, use the name of the element type (in this case, `p`) followed by a period (.) and the element's class name. The result will be as follows:

Hello

Hi

How are you?

Fine. You?

src

Assign the source for an element. This is typically used when you are including an external file or other resource in your site, such as an image, video, audio, CSS, or JavaScript file. For instance, to embed an image in your HTML, use an `img` element with a `src` attribute as follows:

```
<img src="https://edtechbooks.org/content_images/elearning_hacker/rickroll_4k.jpg">
```

href

Assign a link (or horizontal reference) to an element. This is typically used to create a hyperlink to a file or another page.

name

Assign a unique name **string** to an element. This is similar to id, but it is typically used for form elements that will be used to submit information to the server and should be avoided for assigning formatting.

value

Assign a value to an element. This is typically used with input fields like form elements, checkboxes, or radio buttons.

Accessibility Attributes

alt

Provide a description of a visual or audio resource so that screen readers, web crawlers, and those with disabilities can understand the content.

dir

Choose the direction of the text. For example, English would be left-to-right, while Arabic would be right-to-left.

Options include *ltr* for left-to-right and *rtl* for right-to-left.

aria

Accessible Rich Internet Applications (ARIA) values are used in more sophisticated applications where content is dynamically changed. We will not address them here, but if you are dynamically changing content on a page after it is loaded, please realize that you will need to include additional markup to ensure that your materials are accessible.

Table Attributes

Tables have a variety of attributes that can be set, such as cellpadding, cellspacing, width, height, etc., but these formatting attributes should generally be set with CSS instead.

Some common structural attributes that may be used include the following:

colspan and rowspan

Make a cell (`td` or `th`) span across multiple columns or multiple rows:

```
<table>
  <thead>
    <tr><th rowspan="2">Name</th><th colspan="2">Values</th></tr>
    <tr><th>X</th><th>Y</th></tr>
  </thead>
  <tbody>
    <tr><td>Chicago</td><td>1.5</td><td>3.2</td></tr>
  </tbody>
</table>
```

Name	Values	
	X	Y
Chicago	1.5	3.2

Format Attributes

There are many attributes that may be used for formatting content (e.g., `width`, `height`), but these should generally **not be used**. Instead, you should use element- and class-based CSS whenever possible.

If necessary, the style attribute can be used to apply CSS formatting to a single element, but this should be done sparingly and, instead, the CSS for that element should be set in a separate CSS file. This allows formatting to cascade (or be shared) between files and elements and prevents you from having to update formatting for each and every element.

style

Assign [CSS formatting](#) to an element within the HTML tag. This is useful for development and testing, but generally, all CSS formatting should be moved to a separate CSS file and assigned to the element either via the id or class.

Custom Attributes

At times, it can be helpful to store your own metadata in HTML elements. This is best done by creating your own custom attributes. To prevent confusion, you should only create custom attributes that begin with `data-*`. For

instance, if you wanted to store a note about whether an element has been spellchecked or not, you could use `data-spellchecked="true"` .

Cleaning HTML

HTML   Web Development   Code Cleaning

In eLearning development, you will often encounter tools that do a terrible job of converting content to HTML and CSS. This creates all kinds of formatting problems and increases file size for your end users.

For example, here is a bloated HTML snippet from Microsoft Word that occurred just from copying and pasting a single paragraph of text:

```
<p class="MsoNormal"><span style="font-size: 18px; font-family: 'Roboto',serif; mso-ascii-font-
family: 'Times New Roman'; mso-hansi-font-family: 'Times New Roman'; mso-bidi-theme-font: minor-
bidi; mso-themecolor: text1; color: #212529;">Example paragraph <o:p></o:p></span></p>
```

It renders as follows:

> # Example paragraph

Now, here is a cleaner version:

```
<p>Example paragraph</p>
```

And how it renders:

> Example paragraph

Notice that the two versions render the same, but by removing unnecessary elements, we have drastically reduced the size of the HTML elements (from 260 characters to 8, a reduction of 97%) and also reduced the likelihood of formatting errors for our end users and other developers. Also notice that since the original HTML uses the style attribute, a nesting span, and other unnecessary markup, it may not properly assign CSS formatting from the page to the element. This is not an isolated example, and it is common for imported HTML to be bloated by over 90%.

In addition, imported HTML will typically not make use of accessibility features enabled by your web platform, such as adaptive scaling of images to fit different screen sizes.

For these and other reasons, it is good practice to clean HTML that you import from other sources, such as Microsoft Word, into a web authoring platform, such as a Learning Management System or Content Management System.

Remove Styles

Styles can conflict with some CSS properties and also introduce extreme bloat because the formatting is declared on every element instead of just once for the element type or class. Some web authoring platforms have built-in tools to remove styles for you. For instance, in EdTech Books, you can click on `Tools > Code Cleaning > Remove Styles` to remove every style in a document.

Figure 1

Removing Styles in EdTech Books
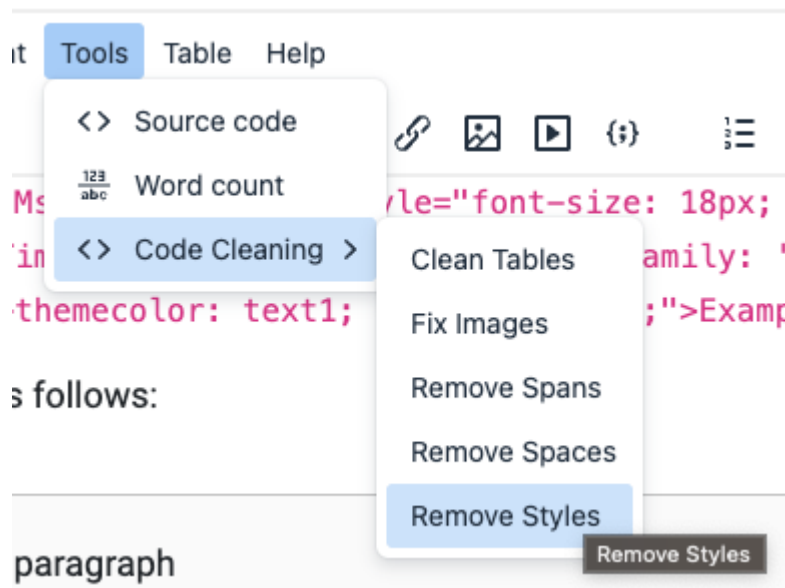
This function uses a simple jQuery command to remove each style attribute as follows, and it could be used in the Inspector Console on other web authoring tools that use jQuery by replacing `chapter-container` with your own encapsulating element id:

```
$('#chapter-container').find('*').removeAttr("style");
```

Alternatively, you can also do this with JavaScript in the Inspector Console as follows by replacing `chapter-container` with your own encapsulating element id:

```
document.getElementById('chapter-container').querySelectorAll('*').forEach(function(element) {
  element.removeAttribute('style');
});
```

You can also achieve this with some text editors that allow for searching and replacing using regular expressions, such as Visual Studio. In the editor, paste your HTML, do a find on the following expression, and replace all results with a blank **string**:
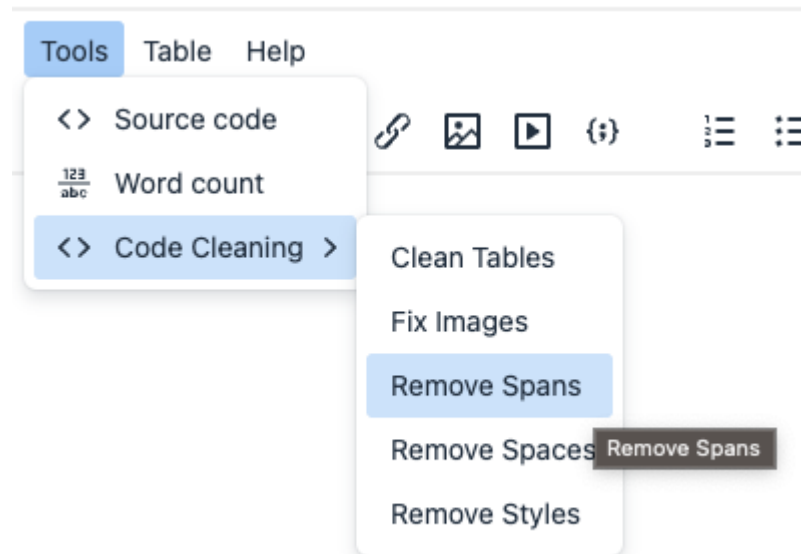
```
style=".*?"
```

Remove Spans

Many tools nest `span` elements within other elements, but these are typically not needed and can really mess up formatting. In EdTech Books, you can click `Tools > Code Cleaning > Remove Spans` to remove every span in a document without losing its contents.

Figure 2

Removing Spans in EdTech Books

This function uses a simple jQuery command to replace each span element with its contents, and it could be used in the Inspector Console on other web authoring tools that use jQuery by replacing `chapter-container` with your own encapsulating element id:

```
$('#chapter-container').find('span').each(function() {
  $(this).replaceWith($(this).contents());
});
```

This can also be achieved using JavaScript as follows:

```
var parent = document.getElementById('parent');
var spanElements = parent.getElementsByTagName('span');
for (var i = spanElements.length - 1; i >= 0; i--) {
  var spanElement = spanElements[i];
  while (spanElement.firstChild) {
    parent.insertBefore(spanElement.firstChild, spanElement);
  }
parent.removeChild(spanElement); }
```

You can also achieve this in a text editor with regular expression support by searching and replacing `<span.*?>` and `</span>` with blank **strings**.

Remove Tool-Specific Elements

Some tools inject their own unique elements into HTML that should be removed. These will vary by tool but can be removed with jQuery as follows, while replacing `chapter-container` with your own encapsulating id and `o\:p` with the name of the tool-specific elements you want to replace:

```
$('#chapter-container').find('o\\:p').each(function() {
  $(this).replaceWith($(this).contents());
})
```

This finds all elements of type `<o:p>`, which is a common Microsoft Word addition, and removes them without losing any content inside them.

You can also do this with JavaScript as follows while replacing `chapter-container` with your own encapsulating id and `o:p` with the name of the tool-specific element.:

```
var parent = document.getElementById('chapter-container');
var wrappedElements = parent.getElementsByTagName('o:p');
for (var i = 0; i < wrappedElements.length; i++) {
  var wrappedElement = wrappedElements[i];
  while (wrappedElement.firstChild) {
    parent.insertBefore(wrappedElement.firstChild, wrappedElement);
  }
  parent.removeChild(wrappedElement);
}
```

You can also achieve this in a text editor by finding and replacing the opening and closing tags with blank **strings**.

Using Artificial Intelligence (AI)

In addition to the above methods, you can also look at the HTML, decide what you want done, and then ask an AI, like ChatGPT, to write a JavaScript function for you. You can then use the function in the Inspector Console.

For example, you might use the following as a prompt:

```
Write a JavaScript function that removes all style attributes within an HTML element with the id
"chapter-container".
```

This content is provided to you freely by EdTech Books.

Access it online or download it at https://edtechbooks.org/elearning_hacker/cleaning_html.

Helpful HTML Snippets

HTML   Web Development   CSS

This page provides some helpful snippets to get you started making common content items. In each example, I provide a Style Method and CSS Method. If possible, the CSS Method is preferred so that you only need to declare the formatting once, but if you are using a web authoring system that does not allow you to edit the CSS, you can revert to the Style Method.

To use the CSS Method, place the first block of markup in the style heading of the document or in your attached CSS file.

Callout Box

Heading
Content
Content

Adaptive Columns

Instead of using tables to create columns or to organize content horizontally, you can use a combination of `<div>` elements with CSS listeners to have columns that adapt to your screen size:

CSS

Web Development   CSS

Cascading Stylesheets (CSS) are ways to define formatting styles that can be applied to HTML elements and shared between HTML pages. This allows you to ensure uniformity across complex HTML projects and to reduce file and download sizes.

| Bootstrap |
| Common CSS Properties |

Bootstrap

CSS   JavaScript   Bootstrap

Bootstrap is a free CSS library that is very common across the internet. It is nice for many reasons, but one thing that it does well is allow you to quickly create well-formatted content and useful snippets that can be used in your learning modules. If Bootstrap is installed by your LMS, then you will have access to many of the formatting options and rich features provided by Bootstrap below (and more).

Basic (CSS-Only) Elements

Alerts

There are no built-in alert elements for HTML, but sites often need some method for designating alerts. Bootstrap does this by assigning `p`, `div`, or other elements with the `alert` class followed by a color formatting class, such as `alert-success` or `alert-danger`.

class="alert alert-primary"

class="alert alert-secondary"

class="alert alert-success"

class="alert alert-danger"

class="alert alert-warning"

class="alert alert-info"

class="alert alert-light"

class="alert alert-dark"

Backgrounds

Backgrounds of elements can be set by changing the class of the elements to one of the following values:

.bg-primary

.bg-primary-subtle

.bg-secondary

.bg-secondary-subtle

.bg-success

.bg-success-subtle

.bg-danger

.bg-danger-subtle

.bg-warning

.bg-warning-subtle

.bg-info

.bg-info-subtle

.bg-light

.bg-light-subtle

.bg-dark

.bg-dark-subtle

.bg-body-secondary

.bg-body-tertiary

.bg-body

.bg-black

.bg-white

.bg-transparent

Buttons

This is a normal HTML button:

Click here

And these are examples of Bootstrap buttons:

btn btn-primary   btn btn-secondary   btn btn-success   btn btn-danger   btn btn-warning   btn btn-info

btn btn-light   btn btn-dark   btn btn-link

Basic formatting for Bootstrap buttons (e.g., size, spacing) is achieved by adding the class `btn` to the button HTML element. Then, you can add a second class to adjust the color, such as `btn-primary` or `btn-secondary`.

Font Colors

Font colors of elements can be set by changing the class of the elements to one of the following values (note that this can also be combined with background classes to give light text a dark background):

.text-primary

.text-primary-emphasis

.text-secondary

.text-secondary-emphasis

.text-success

.text-success-emphasis

.text-danger

.text-danger-emphasis

`.text-warning`

.text-warning-emphasis

`.text-info`

.text-info-emphasis

`.text-light`

.text-light-emphasis

.text-dark

.text-dark-emphasis

.text-body

.text-body-emphasis

.text-body-secondary

.text-body-tertiary

.text-black

`.text-white`

.text-black-50

`.text-white-50`

Margins

Margins are the white space around an element. These values can be set either symmetrically (by referencing the x- and y-axis margins) or asymmetrically (by referencing the top, bottom, start, or end margin) by applying classes to an element as follows:
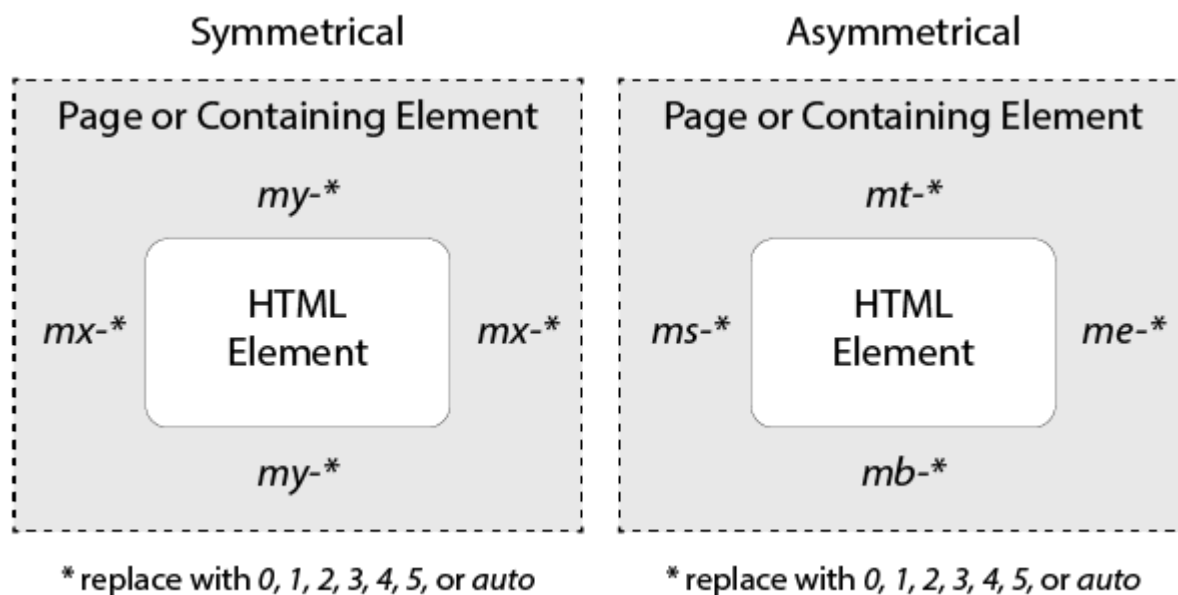
Figure 1

Bootstrap Margin Classes

| Symmetrical | Asymmetrical |

Symmetrical

Page or Containing Element

my-*

HTML Element

mx-*   *mx-**

my-*

* replace with *0, 1, 2, 3, 4, 5,* or *auto*

Asymmetrical

Page or Containing Element

mt-*

HTML Element

ms-*   *me-**

mb-*

* replace with *0, 1, 2, 3, 4, 5,* or *auto*

Symmetrical references are just shorter forms of writing asymmetrical references. So, a class of `mx-2` would be equivalent to a class value of `ms-2 me-2`. The `auto` definition attempts to automatically layout the page, such as by centering the content. You can also apply margins to all four sides of an element with the `m-*` class.

Here are a few examples:

m-1

m-2

m-3

mx-1 my-4

mt-1 me-2 mb-3 ms-4

Padding

Padding is the white space within an element separating an element's content from its own border. These values can be set either symmetrically (by referencing the x- and y-axis padding) or asymmetrically (by referencing the top, bottom, start, or end padding) by applying classes to an element as follows:
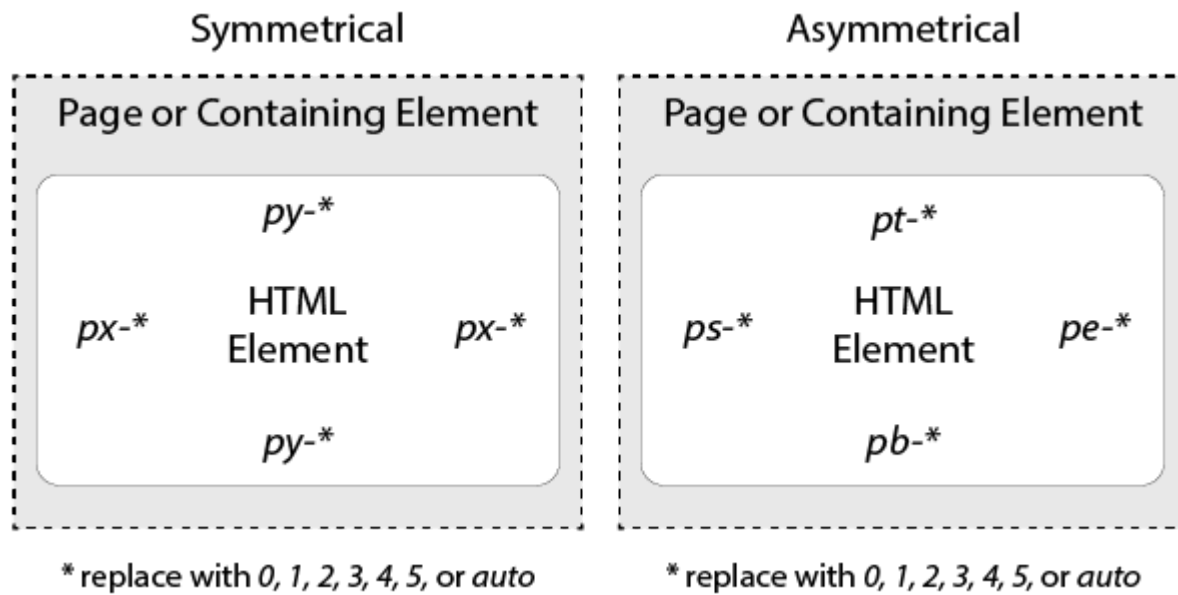
Figure 1

Bootstrap Margin Classes

Symmetrical   Asymmetrical

Symmetrical references are just shorter forms of writing asymmetrical references. So, a class of `px-3` would be equivalent to a class value of `ps-2 pe-2`. The `auto` definition attempts to automatically layout the page, such as by centering the content. You can also apply padding to all four sides of an element with the `p-*` class.

Here are a few examples:

```
p-1
```

```
p-2
```

```
p-3
```

```
px-1 py-4
```

```
pt-1 pe-2 pb-3 ps-4
```

Pro Tip on Padding

When laying out text and other elements, it typically looks better for the x padding to be a bit larger than the y padding. So, if you have a paragraph with a `py-1`, then you'll likely want a `px-2` to go with it.

Progress Bars

Progress bars are normally made in HTML by simply placing one `div` element inside of another, with the outer element serving as the empty background and the inner element serving as the progress indicator. By changing the `width` value

of the inner `div` element's style attribute to a percent of the whole, you can change the visible progress. You can also change the color of either the inner or outer element by changing its class to a background class.
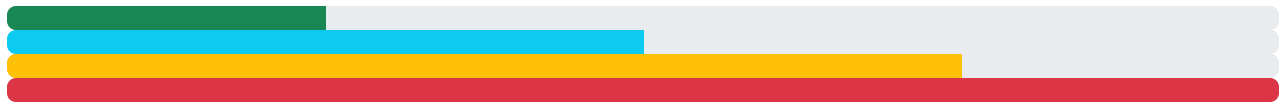
Shadows

Shadows are useful to convey depth to elements on a page, though they should be used intentionally. The shadow classes include `shadow-sm`, `shadow`, and `shadow-lg` and have the following effect when added to an element:

.shadow-sm

.shadow

.shadow-lg

Text Alignment

Content can be horizontally aligned using the following classes: `text-start`, `text-center`, or `text-end`.

.text-start

.text-center

.text-end

Note: Previous versions of Bootstrap used .text-left and .text-right, but since not all languages go from left-to-right, shifting to start and end allows formatting to be better applied across all languages.

Text Size

When resizing text, you should first make sure that you are resizing semantically. So, if you want to resize the text because it is a heading, make it a heading (e.g., H1, H2) rather than just changing the size.

There are cases where you may still want to resize text that is not in a heading. In these cases, you can use the following font-size classes, which correspond to the heading sizes:

.fs-1

.fs-2

.fs-3

.fs-4

.fs-5

.fs-6

Horizontal Layout Elements

Laying out content on a web page horizontally can be problematic, because screen sizes vary and accessibility and machine tools (like screenreaders) may have difficulty understanding the order of content. In the early days of the internet, people often relied on tables to layout content into grids, but this leads to a variety of problems. Instead, current web design tends to use flexible columns and floats to achieve complex horizontal layouts.

Grid Columns

By creating an element in Bootstrap and assigning it the class `row`, Bootstrap will organize all content in that element into 12 possible columns. Then, you can assign the class `col-1`, `col-2`, ... `col-12` to any elements within the row element to have it layout the element as one of the columns.
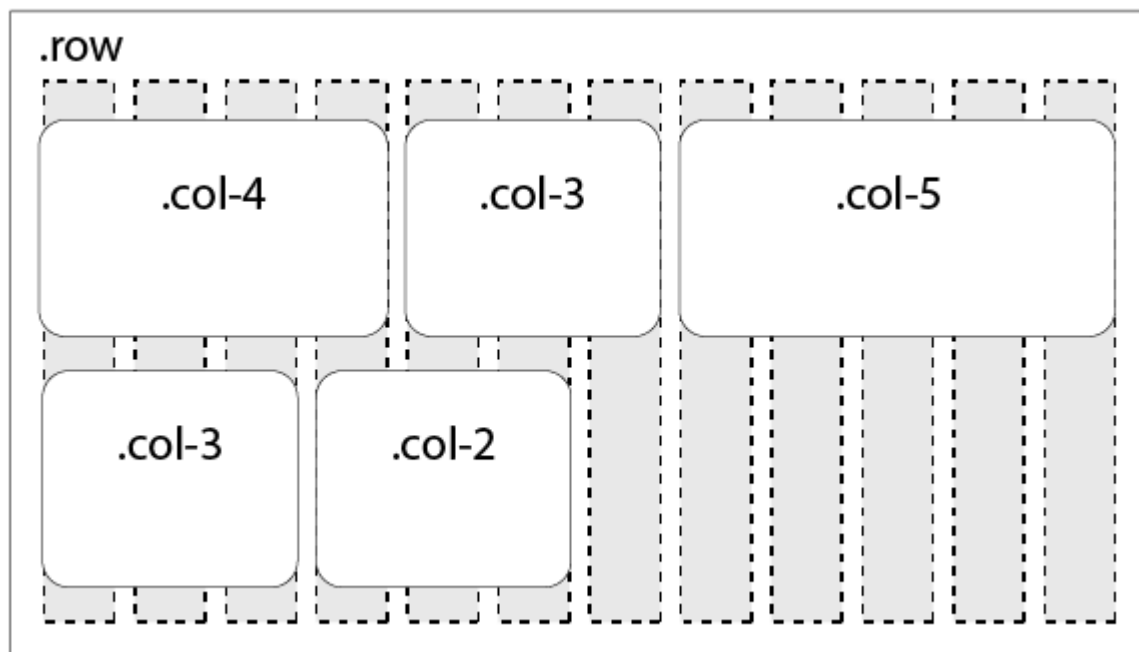
Here is an example of a simple 2-column layout:

```
<div class="row">
  <p class="col-6">Here's the first column.</p>
  <p class="col-6">Here's the second column.</p>
</div>
```

Here's the first column.   Here's the second column.

If we wanted three columns, we could divide the 12 column options however we like between them, such as the following:

```
<div class="row">
  <p class="col-4">Here's the first column.</p>
  <p class="col-4">Here's the second column.</p>
  <p class="col-4">Here's the third column.</p>
</div>
```

Here's the first column.   Here's the second column.   Here's the third column.

Floats

Floats can be used to quickly align an element to the start or end of its parent element and have other elements wrap around it. Floats are most commonly used with images, such as the image floating to the right of this text area.

To apply a float to an element, just add the `float-start` or `float-end` class to it.

Note that elements will continue to wrap around the float until they are told to stop. You can do this by giving the next element the `clearfix` class or by making a new empty element of any kind, such as a div, and giving it the class `clearfix`.

This is a Floating Image

.float-end
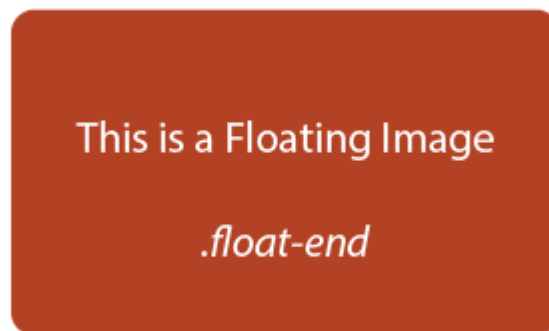
Widths

Bootstrap provides a shortcut for assigning common widths to elements with the `w-25`, `w-50`, `w-75`, and `w-100` classes, which give the element the value width as a percent of the width of the parent element. So, `w-75` would be equivalent to `style="width: 75%;"`. This can be particularly helpful in conjunction with floats by allowing you to define an image as 50% width via `w-50` and floating it at the end of a paragraph via `float-end`.

Flexible Layouts

And since not all screens will be the same size, we need horizontal layouts to react to the width of the screen. Though we might want an image to float to the right on a large screen, we might want the text wrapping to go away on a small screen (so that we have more than two words of text per line.

To achieve this, we can include references to the user's current screen size to any of the layout elements above to apply it to only certain screen sizes, such as `sm`, `md`, `lg`, `xl`, and `xxl`. These are called breakpoints. Because modern web practices have a mobile-first mindset, classes will apply in a manner that assume you are applying formatting at the smallest possible screen size unless you explicitly apply it only to a larger screen size. So, `md` will also apply to screen widths of `lg`, `xl`, and `xxl`, but `xxl` will not apply to any other screen widths.
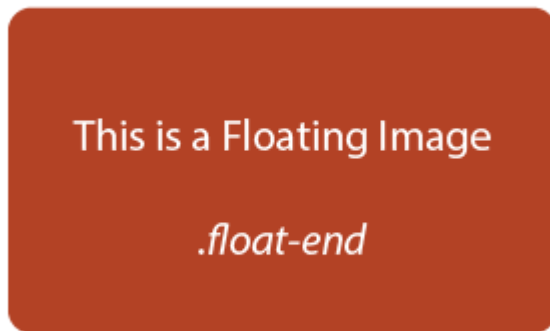
So, if you apply a class of `float-xl-end`, then the element will float only on screen widths of `xl` or larger. You can test this in your browser by resizing this page and noticing that the floating image to the right no longer floats once the screen gets smaller. Contrast this with the floating image above that always floats no matter the screen size.

These width modifiers can be applied to the other horizontal layout elements above to achieve very powerful layouts. For instance, you could apply `col-12 col-md-6` to an element to make it six columns wide on `md` and larger screens, and 12 columns wide on smaller screens. Or, you could make an image 50% width on larger screens and 100% width on smaller screens by assigning it the classes `w-100 w-lg-50`.

Display

You can also decide to completely hide or show content based on the user's screen size by using the `d-*` (or display) classes. Some of the more common uses of this are to alternate between `d-block` (to have an element fill the screen horizontally) and `d-none` (to hide the element). For instance, by default a `p` element is using a block display, so applying `d-block` to it won't change anything, but if you give a `p` element a `d-none` class, then it will be hidden.

You can combine these classes with the breakpoints above to have elements show or hide themselves depending upon the size of the screen. So, an alert message with classes `d-none d-md-block` will only reveal itself once the screen reaches the `md` breakpoint size. Or, an image with classes `d-lg-none` will hide itself once the screen reaches the lg breakpoint.

You can also have elements only show themselves under certain conditions, like if the page is being printed. For instance, an image with the class `d-print-none` won't be visible when you print the page, while a paragraph with the class `d-none d-print-block` will only be shown when the page is being printed.

Advanced (CSS + JS) Elements

Accordion

Accordion Item #1   ⌄

This is the first item's accordion body. It is shown by default, until the collapse plugin adds the appropriate classes that we use to style each element. These classes control the overall appearance, as well as the showing and hiding via CSS transitions. You can modify any of this with custom CSS or overriding our default variables. It's

also worth noting that just about any HTML can go within the `.accordion-body`, though the transition does limit overflow.

Accordion Item #2 ⌄

This is the second item's accordion body. It is hidden by default, until the collapse plugin adds the appropriate classes that we use to style each element. These classes control the overall appearance, as well as the showing and hiding via CSS transitions. You can modify any of this with custom CSS or overriding our default variables. It's also worth noting that just about any HTML can go within the `.accordion-body`, though the transition does limit overflow.

Accordion Item #3 ⌄

This is the third item's accordion body. It is hidden by default, until the collapse plugin adds the appropriate classes that we use to style each element. These classes control the overall appearance, as well as the showing and hiding via CSS transitions. You can modify any of this with custom CSS or overriding our default variables. It's also worth noting that just about any HTML can go within the `.accordion-body`, though the transition does limit overflow.

Common CSS Properties

Cascading Style Sheet (CSS) property options have expanded continually over the years so that there are now hundreds to choose from. For a complete list, see W3Schools. However, like HTML tags, there are only about a dozen that you will use regularly. In this chapter, we'll just apply these to the `style` attribute of an HTML element to see their effects.

background

Set a background color using a common color name:

```
<p style="background: pink;">An example paragraph.</p>
```

An example paragraph.

Set a background color using rgba notation:

```
<p style="background: rgba(200,200,200,1);">An example paragraph.</p>
```

An example paragraph.

Set a background color using hexadecimal notation:

```
<p style="background: #e5e5e5;">An example paragraph.</p>
```

An example paragraph.

color

Set the font color of text using a common color name:

```
<p style="color: blue;">An example paragraph.</p>
```

An example paragraph.

Set the font color of text using rgba notation:

```
<p style="color: rgba(40, 40, 220, 1);">An example paragraph.</p>
```

An example paragraph.

Set the font color of text using hexadecimal notation:

```
<p style="color: #2222ee;">An example paragraph.</p>
```

An example paragraph.

font-family

Set the font of the element:

```
<p style="font-family: Courier, 'Times New Roman'; background: pink;">An example paragraph.</p>
```

```
An example paragraph.
```

Please note that the `font-family` property can include multiple values separated by a comma. This is done so that if a user's browser does not have the desired font installed, it will revert to using the next identified font. Also notice that *Times New Roman* is in a single quote. This is done because that font name has spaces within it, which would otherwise confuse the value.

height and width

Set the size of the element:

```
<p style="height: 80px; background: pink;">An example paragraph.</p>
```

An example paragraph.

```
<p style="height: 80px; width: 50%; background: pink;">An example paragraph.</p>
```

An example paragraph.

float

Set the alignment of an element on the page. This is useful for creating columns, inset images, and so forth.

Float an image to the right of a block of text:

```
<img src="https://edtechbooks.org/content_images/elearning_hacker/rickroll_4k.jpg" style="width:
50%; float: right;" class="img-fluid">
<p>An example paragraph. An example paragraph. An example paragraph. An example paragraph. An
example paragraph. An example paragraph. An example paragraph. An example paragraph.</p>
<p style="clear: both;"></p>
```

An example paragraph. An example paragraph. An example paragraph. An example paragraph. An example paragraph. An example paragraph. An example paragraph. An example paragraph.

Please note that the float will not stop on its own. Rather, you need to include an element after the floated object that includes the property-value pair `clear: both;` to prevent subsequent elements from being affected by the float. Options include *left* or *right*.

padding

Padding represents the amount of space between the edge of an element and the text or other content inside it.

Set the padding of all four edges:

```
<p style="padding: 40px; background: pink;">An example paragraph.</p>
```

An example paragraph.

Set the padding of the y-edges and x-edges separately:

```
<p style="padding: 20px 40px; background: pink;">An example paragraph.</p>
```

An example paragraph.

Set the padding of each edge separately, moving clockwise:

```
<p style="padding: 10px 20px 30px 40px; background: pink;">An example paragraph.</p>
```

An example paragraph.

margin

Margin represents the amount of space between an element and its surrounding or parent element.

Set the margin of all four edges:

```
<p style="margin: 40px; background: pink;">An example paragraph.</p>
```

An example paragraph.

Set the margin of the y-edges and x-edges separately:

```
<p style="margin: 20px 40px; background: pink;">An example paragraph.</p>
```

An example paragraph.

Set the margin of each edge separately, moving clockwise:

```
<p style="margin: 10px 20px 30px 40px; background: pink;">An example paragraph.</p>
```

An example paragraph.

text-align

Set the horizontal alignment of an element:

```
<p style="text-align: center;">An example paragraph.</p>
```

An example paragraph.

Options include *left, right, center,* and *justify*.

border

Set the border size, color, and type of the element:

```
<p style="border: 1px black solid;">An example paragraph.</p>
```

An example paragraph.

Do the same with a larger, dashed border:

```
<p style="border: 4px pink dashed;">An example paragraph.</p>
```

An example paragraph.

Please note that borders are heavily overused and introduce unnecessary complexity and cognitive load. So, only use borders when they are necessary.

border-radius

Make all the corners of the element rounded:

```
<p style="border-radius: 10px; padding: 8px 16px; height: 80px; background: pink;">An example paragraph.</p>
```

An example paragraph.

Make each corner rounded in different amounts, moving clockwise:

```
<p style="border-radius: 10px 20px 30px 0; padding: 8px 16px; height: 80px; background: pink;">An example paragraph.</p>
```

An example paragraph.

JavaScript

JavaScript

JavaScript is a common programming language that is used ubiquitously across the web. You can use it for many things, from creating dynamic website content to scraping and organizing data.

JavaScript Listeners
JavaScript Basics
JSON
jQuery
REST APIs

JavaScript Listeners

JavaScript

JavaScript is normally run on your page in one of three ways. First, if the script is written directly in a `<script>` HTML element or an attached JavaScript file, then it will run when it is loaded. So, the following script would run when a page is opened or refreshed:

```
alert("The page was reloaded.");
```

The second way is for a function to be called. So, let's say that we include the following script at the top of a page:

```
function CreateMyOwnPopup(){
  alert("Here's my popup.");
}
```

This script will register the `CreateMyOwnPopup` function, but the page won't actually do anything until something calls the function, such as if we included the following later in the page:

```
CreateMyOwnPopup();
```

And the third way that JavaScript is typically run on a webpage is in conjunction with a listener. Listeners are registered when the page is loaded, and then they run the script inside them when they hear the listener being activated. For instance, the following script would find a button with the id `button1` and run the `CreateMyOwnPopup` function any time the button is clicked:

```
document.getElementById('button1').addEventListener('click', function() {
  CreateMyOwnPopup();
});
```

In this case, JavaScript registers a listener on the `button1` element to do something every time the click event occurs (i.e., someone clicks on it). See below for this example in action:

Additionally, jQuery has a shorter way of adding the same shortcut as follows:

```
$('#button1').on('click', function() {
  CreateMyOwnPopup();
}
```

JavaScript can listen for many different types of events, and available events can vary based on the element you are listening to, but some common examples include `click`, `change`, `keyup`, `keydown`, and `scroll`.

JavaScript Basics

JavaScript

Mathematics

Like any programming language, JavaScript understands mathematical notations and uses operators like `+` (addition), `−` (subtraction), `*` (multiplication), and `/` (division) to manipulate numbers.

In your Inpector Console, use JavaScript to do some arithmetic. Who knew that your browser was a fully-fledged calculator?!

Variables

Variables are useful for storing data to be retrieved or manipulated later. To declare a variable, just type `var` followed by the name you'd like to assign the variable, then `=` and the value. For instance, if you wanted to create a variable called `year` and assign it the value of `1999`, you could do so as follows:

```js
var year = 1999
```

Or if you wanted to create a variable called course and assign it the value of IPT660, you could do so as follows:

```js
var course = "IPT660"
```

Note that if your value is textual, then you need to wrap it in quotes, but numbers don't need quotes.

After you have done this, if you ever type the word `year` into the console, it will return `1999`, and if you type `course`, it will return `IPT660`.

The word `var` is only used when we first declare a variable. Once the variable has been declared, you can then change it if needed. So, if we wanted to change the value of the course variable to `IPT531`, we'd just do the following:

```js
course = "IPT531"
```

Now, if we type the word `course`, it will return `IPT531` instead of `IPT660` (which is now totally forgotten).

Logical Operators

In addition, JavaScript also can logically compare values with `<` (less than), `<=` (less than or equal to), `<` (greater than), `>=` (greater than or equal to), `==` (equal to), `!=` (not equal to), `&&` (and), and `||` (or). (Remember that the equals sign `=` does not compare values, because it is rather used in JavaScript to assign variables.)

In your Inspector Console, if you type `1<2`, it will return `true`, while `1>2` will return `false`. You can combine these with the mathematical functions to compare complex values, such as 272*331>254*352.

Is this statement true or false `272*331>254*352` ?

	True
	False

Logical operators are not confined to numbers; they can also be used for comparing more complex pieces of data, such as text and objects. For instance, `"cheese"=="crackers"` would return `false`, while `"cheese"!="crackers"` would return `true`.

You can also combine variables with the mathematical and logical operators. For instance, if you type `year + 10`, the console will return `2009`.

Logical Statements

You can use logical operators in conjunction with logical statements like `if`, `else`, and `while` to tell JavaScript what to do under different circumstances and for how long. For instance, since we declared the value of the `year` variable to be `1999`, we can ask JavaScript to give us a warning if the course was taken longer than 10 years ago as follows:

```
var currentYear = 2023;
if(year<currentYear-10) {
  alert("The course is too old.");
} else {
  alert("The course is good!");
}
```

Functions

Functions are actions that you can store and run on data or objects many times. These are declared similar to variables and can then be referenced by their name and fed variables to run. For instance, let's take our previous statement and wrap it in a function called `CheckYear` as follows:

```
function CheckYear(year) {
  var currentYear = 2023;
  if(year<currentYear-10) {
    alert("The course is too old.");
  } else {
    alert("The course is good!");
```

```
   }
}
```

Once you have done this, you can now call CheckYear() and feed it any value you like to see if a year is too old as follows:

```
CheckYear(2020)
```

You may do this with any value:

```
CheckYear(1830)
```

JSON

JavaScript   JSON

JavaScript Object Notation (JSON) is a lightweight way to transfer JavaScript object data between a variety of apps and tools as a simple **string**. It uses a simple notation consisting primarily of braces `{ }`, brackets `[ ]`, commas, and quotes to construct potentially complex data structures.

Objects

JavaScript objects are encapsulated in braces `{ }` as name and value pairs separated with commas. For example, a simple user object might consist of the following:

```
{
  firstName: "Jane",
  lastName: "Doe",
  email: "janedoe@email.com"
}
```

Arrays

JavaScript arrays are encapsulated in brackets `[ ]` as values separated with commas. For example, a simple list of names might consist of the following:

```
[
  "Jane Doe",
  "John Doe",
  "Teddy Ruxpin"
]
```

Structuring Data

Objects and arrays can be nested inside one another to create nested data structures, such as in this list of student course schedules:

```
[
    {
```

```
        name: "Jane Doe",
        email: "janedoe@email.com",
        courses: [
            {
                name: "Biology",
                section: "1"
            },
            {
                name: "Chemistry",
                section: "2"
            }
        ]
    },
    {
        name: "Teddy Ruxpin",
        email: "teddy@email.com",
        courses: [
            {
                name: "Philosophy",
                section: "1"
            },
            {
                name: "Chemistry",
                section: "1"
            }
        ]
    }
]
```

Note that this example consists of an array of user objects that each has an array of course objects.

Based on this schedule, would Jane and Teddy share any course sections?

	Yes
	No

Retrieving Data

To retrieve a data value from an object, you would reference the object and then the name of the key in the key value pair you desire separated with a period. For instance, we could assign a list of users to a variable as follows:

```
var user = {
  firstName: "Jane",
  lastName: "Doe",
  email: "janedoe@email.com"
};
```

Try this in your Inspector Console by pasting the above code and hitting enter. This sets the user variable in your console. Then, if we wanted to return the firstName of the user, we could simply type `user.firstName` , which would

return "Jane".

Alternatively, you can also access object values by using brackets and quotation marks, such as `user["firstName"]`.

To retrieve data from an array, you would reference the array and then the desired item's order number in the array (starting from zero). So, we could assign a list of names to an array as follows:

```
var userNames = [
  "Jane Doe",
  "John Doe",
  "Teddy Ruxpin"
]
```

And then we could reference each element by typing `userNames[0]` (which would return "Jane Doe"), `userNames[1]` (which would return "John Doe"), etc.

In the case of structured data, like our example above, we can reference data anywhere in the structure by following this pattern. Try pasting the following into your Console:

```
var studentSchedules = [
    {
        name: "Jane Doe",
        email: "janedoe@email.com",
        courses: [
            {
                name: "Biology",
                section: "1"
            },
            {
                name: "Chemistry",
                section: "2"
            }
        ]
    },
    {
        name: "Teddy Ruxpin",
        email: "teddy@email.com",
        courses: [
            {
                name: "Philosophy",
                section: "1"
            },
            {
                name: "Chemistry",
                section: "1"
            }
        ]
    }
];
```

Then, retrieve Teddy's email address by typing `studentSchedules[1].email`.

How would you retrieve the name of Jane's second course?

- [ ] studentSchedules[0].courses[1].name

- [ ] studentSchedules[0].name

- [ ] studentSchedules[0].courses.name

jQuery

JavaScript   jQuery

[jQuery](#) is a free JavaScript library that may be optionally installed on a webpage to provide additional functionality. For our purposes, jQuery is especially useful for detecting, changing, or extracting structured content from a webpage.

The simplest way to determine if a webpage has jQuery installed is to run the following command in the Inspector Console:

`$()`

The dollar sign is jQuery's shorthand method for accessing jQuery functions. If this does not work, you can also try the following:

`jQuery()`

If either of these commands doesn't return an error, then you can run jQuery commands on the page. The `$()` operator is just a shorthand version of `jQuery()`, so if `$()` is not available but `jQuery()` is, you can modify any command to use `jQuery()` instead of `$()`. You can also [install jQuery on your own page or applications](#).

Including jQuery in Your Project

The simplest way to include jQuery in your project is via a CDN by appending the following JavaScript to the end of your document (just before the `</body>` tag):

```
<script src="https://code.jquery.com/jquery-3.7.1.min.js"
integrity="sha256-/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo=" crossorigin="anonymous"></script>
```

By placing it at the end, we allow the content on the rest of the page to be loaded and displayed prior to jQuery being fetched and downloaded, speeding up the user's access to the content.

Querying the DOM

jQuery helps us to navigate the [DOM](#) more quickly and to find the elements or content we want. It does this by allowing us to have the DOM return elements that match a particular query. Queries can use element names, attributes, and other element features to find what we want. For example, if we wanted to return a list of all links (`<a>`) on a web page, we could just run the following:

```
$("a")
```

We can then use jQuery to dig into these elements further or to run functions on each element. So, if we wanted to return just the text of all links on a page, we could run the following:

```
$("a").text()
```

Or if we wanted to send the text of each link to the console, we could do the following:

```
$("a").each(function() { console.log($(this).text()) })
```

We can also use jQuery to change the contents of a webpage. For instance, if we wanted to make the text of all links on the page read "LINK HERE", we could run the following:

```
$("a").each(function() { $(this).text("LINK HERE") })
```

Attributes

You can select elements by whether or not they have particular attributes or whether their attribute values match your query. For instance, you can return all images that have an `alt` attribute defined as follows:

```
$("img[alt]")
```

You can also return all images that have a blank alt attribute as follows:

```
$("img[alt='']")
```

Shorthand Attribute Selectors

Perhaps the most common attributes you will use to select elements is the `id` and `class` attribute. For this reason, jQuery has a shorthand way to accessing classes similar to CSS, where the class is identified with a period and the id is identified with a hashtag.

So, if we wanted to select all `<p>` elements with the class "callout", we could use `$("p.callout")`, which is shorthand for `$("p[class='callout']")`, which is shorthand for the pure JavaScript `document.querySelectorAll('p.callout')`.

Or if we wanted to select the `<p>` element with the ID "paragraph1", we could use `$("p#paragraph1")`, which is shorthand for `$("p[id='paragraph1']")`, which is shorthand for the pure JavaScript `document.querySelectorAll('p#paragraph1')`.

Negative Selectors

You can also append `:not` to a query in order to choose the opposite. For instance, you can choose all images that do not have an `alt` attribute defined as follows:

```
$("img:not([alt])")
```

Nested Elements

You can find nested elements by including a space between query operators. For instance, if you wanted to find all links that are inside paragraphs, you could run the following:

```
$("p a")
```

Or if you wanted all `<h3>` elements inside `<div>` elements that have the `callout` class, you could run the following:

```
$("div.callout h3")
```

REST APIs

JavaScript   API   JSON

A REST API is a Representational State Transfer Application Programming Interface. That may sound daunting, but let's break it down in an eLearning scenario. "Representational State" means that you are taking some piece of content and creating a snapshot of it in time. "Transfer" means that you are taking that snapshot and moving it to a new location on the internet. "Application Programming Interface" means that you are using the new location's programming interface to tell it what to do with the snapshot once it's transferred. So, a REST API simply means that we are taking a snapshot of some piece of data and transferring it to a new location on the internet in a way that the new location can understand what to do with it.

Pull vs. Push

Pull means "give me your data so I can do something with it." So, a pull REST request in an eLearning scenario might be asking a content provider "give me your content so I can put it in my module" or asking a tool "give me my student's grade in your system."

Push means "take my data and do something with it." So, a push REST request in an eLearning scenario might be asking a gradebook "take this grade and add it to the student's gradebook" or asking a Learning Record Store "take this user xAPI data and store it for later analysis."

Security

Since most REST API requests involve multiple sites, there is often a need to ensure that the data being sent is coming from and going to the intended audience without being tampered with. This is typically done by including an API key or other security measure. Please note that when using an API key in client-side JavaScript that anyone can see your key in the inspector.

Data Format

When using JavaScript for REST API requests, the most common data format is JSON, which can then be parsed into JavaScript objects and manipulated on a web page, such as by placing it in an HTML element.

Examples

Pull Example

You can pull content directly from any EdTech Book publication into your eLearning module by using the ETB API. Here is an example of a JavaScript REST call that brings in the content of a chapter with ID 155:

Many content providers and other services have APIs that you can leverage, but most will require some form of secure transfer or login. Some examples include the following:

- Library of Congress
- RapidAPI Hub
- Hugging Face

Push Example

You can push content to EdTech Books, such as xAPI experience data, by using the ETB API. Note that the push API requires an api_key, which you may find in your user settings.

Canvas

Canvas

Canvas Icons and Buttons

Canvas Icons and Buttons

Canvas

Canvas seems to use a modified version of [Bootstrap](#) and other libraries that you can tap into when creating content. Some of the most helpful ways this can assist your course designs is to give you access to icons and buttons.

General Use

Canvas has a set of built-in icons that you can find documented in their [iconography](#). These icons are set to content in a course by setting classes on content. The simplest way to include an icon in your content is to use the `i` attribute followed by the icon type class and the icon name class.

Icon Type Classes

There are two icon type classes: `icon-Solid` and `icon-Line`.

Icon Name Classes

There are many icon names that you can find on the [iconography](#) page.

For example, to include a solid-style alerts icon, simply paste the following line to your content:

```
<i class="icon-Solid icon-alerts"></i>
```

Or to include a line-style audio icon, use the following:

```
<i class="icon-Line icon-audio"></i>
```

Buttons

The Canvas editor may not allow you to create `button` HTML elements, but you can still create `a` HTML elements and make them look like buttons by applying button classes.

```
<a class="Button">Button</a>
```

```
<a class="Button Button--Primary">Button</a>

<a class="Button Button--secondary">Button</a>

<a class="Button Button--success">Button</a>

etc...
```

Button

Button Button--primary

Button Button--secondary

Button Button--success

Button Button--warning

Button Button--danger

Button Button--link

> Note that colors may differ depending upon your institution's styles.

Button Sizes

To change button sizes, you can also add the following classes: `Button--large`, `Button--small`, and `Button--mini`.

Button--large

Button--small

Button--mini

Buttons and Icons Together

To use buttons and icons together, you can simply apply icon classes to the button a element as in these examples:

```
<a class="Button Button--danger icon-Line icon-audio" type="button">Hey, this is a red button with
an audio icon!</a>
```



```
<a class="Button Button--primary icon-Line icon-analytics">Hey, this is a primary button with an
analytics icon!</a>
```



```
<a class="Button Button--success icon-Solid icon-check Button--large">Hey, this is a big 'ol success
button with a check icon!</a>
```

Web Scraping

Legalities and Ethics
Extracting HTML Elements from a Web Page using JavaScript
Extracting Structured Content with an API and JavaScript

Legalities and Ethics

Ethics

■ ■ ■

Prior to downloading, parsing, or using the content of a website in any way, you should always consider the legality and ethics of what you are doing. From a legal perspective, there are at least three things to consider:

1. anti-hacking and cyber crime laws;
2. copyright laws;
3. and terms of use.

Anti-hacking and cybercrime laws vary from country to country and state to state. Most of these laws regulate unauthorized access to information systems or the improper use of such systems. Merely parsing content from a site would not typically constitute a crime, but if you copied, stored, shared, or used data in ways that the owner did not permit, then these laws might come into play. If you have questions about this, please consult a legal expert in cybercrime law.

If you are making copies of content (e.g., extracting data from a site and storing it on your computer), then copyright laws come into play. Each country has its own copyright laws, but in the U.S., you will generally either need permission from the content owner or ensure that your use constitutes Fair Use under the law. If you have questions about this, please consult a legal expert in copyright law.

Finally, the Terms of Use is a statement typically found on a site where the content owner describes what constitutes appropriate use of their site. Note that owners can claim anything they want in a Terms of Use statement (even things that they are not legally allowed to claim) and that the user is typically legally bound to the terms if they have been given adequate notice and accept them. Even without this, however, the Terms of Use statement allows you to understand how the owner sees and may seek to enforce their rights, and you should generally follow the requirements outlined in the Terms of Use statement. If you have questions about this, please consult a legal expert.

In addition to legal considerations, you should also consider the ethics of what you are doing and ensure that your behavior does not harm people or organizations.

Robots.txt Files

In addition to the above, most sites also have a robots.txt file at their root level to communicate to automated tools which areas of the site should and should not be scraped, indexed, etc. For instance, if you go to https://edtechbooks.org/robots.txt in your web browser, you will find a list of areas of this website that may and may not be indexed. You can find such documents on just about any site by typing /robots.txt after the main domain name.

Such documents are helpful to guide you in determining which areas of a site the developer wants you to access via automated methods and which areas they deem to be off-limits.

Extracting HTML Elements from a Web Page using JavaScript

HTML   JavaScript   Web Scraping   jQuery

In your Inspector Console, you can run JavaScript on any web page to extract information from it, but this requires looking for patterns and telling JavaScript what information you want. Prior to doing this, you should ensure that your activities are legal and ethical.

jQuery vs. Pure JavaScript

As I provide code examples, I will primarily use jQuery. However, not all web pages have jQuery installed, and so the provided jQuery commands may not work on every web page. For this reason, I've also provided pure JavaScript for many commands and any jQuery commands can be converted to pure JavaScript if needed.

Looking for Patterns and Identifiers

On the web page, find a piece of information that you would like to extract, right-click it, and view it in the Inspector. Notice the following about the information:

1. What type of element is it in (e.g., `<p>`, `<span>`, `<div>`)?
2. What is the structure of its parent and children elements?
3. Does it or its immediate parent have a `class` attribute assigned?

Most of the time, this is all the information you need to identify other elements like the one you are inspecting. For instance, right-click and inspect the following email address:

admin@edtechbooks.org

Notice the following:

1. It is a link.
2. Its immediate parent is a `<p>` element.
3. It does not have a class, but its parent has the class `email-address`.

Now, in the Inspector Console, run the following jQuery command:

```
// jQuery
$('.email-address').each(function() {
    console.log($(this).text());
});

// JavaScript
document.querySelectorAll('.email-address').forEach(function(e) {
    console.log(e.innerText);
});
```

Notice that this gives you the email address in the console, but it also returned two other email addresses it found on the page that follow the same pattern (in this case, they all are in `<p>` elements that have the `email-address` class).

Let's try this with something else. Let's return all list items on the page by running the following:

```
// jQuery
$('#chapter-container li').each(function() {
    console.log($(this).text());
});

// JavaScript
document.querySelector('#chapter-container').querySelectorAll('li').forEach(function(e) {
    console.log(e.innerText);
});
```

This command gets the page's element with the id `chapter-container` and returns all list item elements (`<li>`) within it, exporting the text of each element to the Inspector Console.

Try this with headings as follows:

```
// jQuery
$('h2').each(function() {
    console.log($(this).text());
});

// JavaScript
document.querySelectorAll('h2').forEach(function(e) {
    console.log(e.innerText);
});
```

This returns all `<h2>` elements on the page.

How about all links on the page?

```
// jQuery
$('#chapter-container a').each(function() {
    console.log($(this).attr('href'));
});

// JavaScript
var links = document.querySelector('#chapter-container').querySelectorAll('a');
links.forEach(function(link) {
    console.log(link.getAttribute('href'));
});
```

Notice with this one instead of using `.text()` we used `.attr('href')` to return the value of the `href` attribute instead of the displayed text.

Now, let's try one more thing and collapse all of our results together into a single comma-separated list. To do this with headings, we can run the following command:

```
// jQuery
var headings = new Array();
$('h2').each(function() {
    headings.push($(this).text());
});
console.log(headings.join(','));

// JavaScript
var headings = document.querySelectorAll('h2');
var headingTexts = Array.from(headings).map(function(heading) {
    return heading.textContent.trim();
});
console.log(headingTexts.join(', '));
```

In this example, we first created an array to capture the results, then added each result to it, and then output the results to the console as a **string** where all elements were joined together with a comma.

Activity: Extract All Emails as a Comma-Separated List

Using the previous examples as a guide, output all email addresses on the page into a comma-separated list in the Inspector Console.

Helpful Examples

ERIC Search Results

For this example, let's say that you are doing a literature review on the topic of "social media," and you want to get all results from an ERIC search results page on the topic. First, access the search results normally by going to https://eric.ed.gov/?q=social+media

Figure 1

ERIC Results for "Social Media"

Then, in the Inspector, you can run a jQuery command to get all of the article names as follows:

```
$('.r_t').each(function() {
    console.log($(this).text());
});
```

Or, if you want the direct links for all articles, try the following:

```
$('.r_f a').each(function() {
    console.log($(this).attr('href'));
});
```

Or, if you only want the direct links to articles that have a full-text download, try the following:

```
$('.r_f a').each(function() {
    if($(this).attr('href').includes('fulltext')) console.log($(this).attr('href'));
});
```

This process works because (a) each article title on the site is inside an element that has a class called "r_t," (b) each direct link is within an element that has a class called "r_f," and (c) each full-text direct link goes to a file that has the phrase "fulltext" in the URL.

Though this is all neat, the results aren't super helpful just yet, because you probably want to take them to a spreadsheet or somewhere else where you can manipulate them. Again, JavaScript can help us by structuring the data we want into an array and then displaying it in a table in the Inspector like this:

```
var results = [];
$('.r_t').each(function(i)
{results.push({'title':$(this).text(),'authors':$('.r_a').eq(i).text(),'abstract':$('.r_d').eq(i).te
```

```
xt(),'href':$('.r_f a').eq(i).attr('href'),})}]);
console.table(results);
```

Figure 2

Tabular Results of an ERIC Search

(index)	title	authors	abstract	href
0	'Social Media Takeover: Usi…	'Garcia, Nancy; Brooks, Mar…	'The opportunities in socia…	'?redir=http%3a%2f%2fdx.doi…
1	'Getting Social: Postgradua…	'Le Busque, Brianna; Mingoi…	"Social media are becoming …	'?redir=http%3a%2f%2fdx.doi…
2	'The Mediating Role of Life…	'Yilmaz, Mustafa; Erduran T…	'The coronavirus pandemic h…	'http://files.eric.ed.gov/f…
3	"Rural Elementary Teachers'…	'Hensley, Matt; Waters, Ste…	'The use of social media co…	'?redir=http%3a%2f%2fdx.doi…
4	'Reducing Problematic Socia…	'Stinson, Lesleigh; Dallery…	'Problematic social media u…	'?redir=http%3a%2f%2fdx.doi…
5	'Types of Social Media Use …	'Winstone, Lizzy; Mars, Bec…	'Evidence on how different …	'?redir=http%3a%2f%2fdx.doi…
6	'Measuring the Effect of So…	'Nurudeen, Mohammed; Abdul—…	'With the advent of smartph…	'?redir=http%3a%2f%2fdx.doi…
7	"Exploring Parents' Intenti…	'Douglas, Kimberly D.; Smit…	'Social media may promote h…	'?redir=http%3a%2f%2fdx.doi…
8	'Social Media and Ethical B…	'Floress, Margaret T.; Cono…	'Today, social media is gai…	'?redir=https%3a%2f%2fwww.n…
9	'Urgency of Social Media—Ba…	'Suastika, I. Nengah; Suart…	'This study aimed at formul…	'http://files.eric.ed.gov/f…
10	'Exploring the Role of Soci…	'Bell, Ashley Rebecca; Tenn…	'Social media platforms rel…	'?redir=http%3a%2f%2fdx.doi…
11	'Student—Professor Social M…	'Cistulli, Mark D.; Snyder,…	'This article explores soci…	'?redir=http%3a%2f%2fdx.doi…
12	'Relationship between Socia…	'Farid, Muhammad Faisal; Ja…	'Social media propagation c…	'http://files.eric.ed.gov/f…
13	'Social Media as Inadverten…	'Kotsonis, Alkis — Journal …	'My aim in this paper is to…	'?redir=http%3a%2f%2fdx.doi…
14	'Social Media Usage and Aca…	'Hameed, Irfan; Haq, Mirza …	'Purpose: Social media has …	'?redir=http%3a%2f%2fdx.doi…

Extracting Structured Content with an API and JavaScript

JavaScript   Web Scraping   API

Application Programming Interfaces (APIs) are tools provided by developers to give other developers access to content or software features in more direct ways than going through a web interface or HTML. As Figure 1 illustrates, when it comes to scraping content, APIs have two advantages over parsing web pages. First, APIs can return structured data that is already parsed, allowing you to avoid finding patterns in the rendered HTML, and second, APIs can sometimes return data that is not available in the HTML.

Prior to using APIs, you should ensure that your activities are legal and ethical.

Figure 1

Accessing Information via a Web Page vs. an API
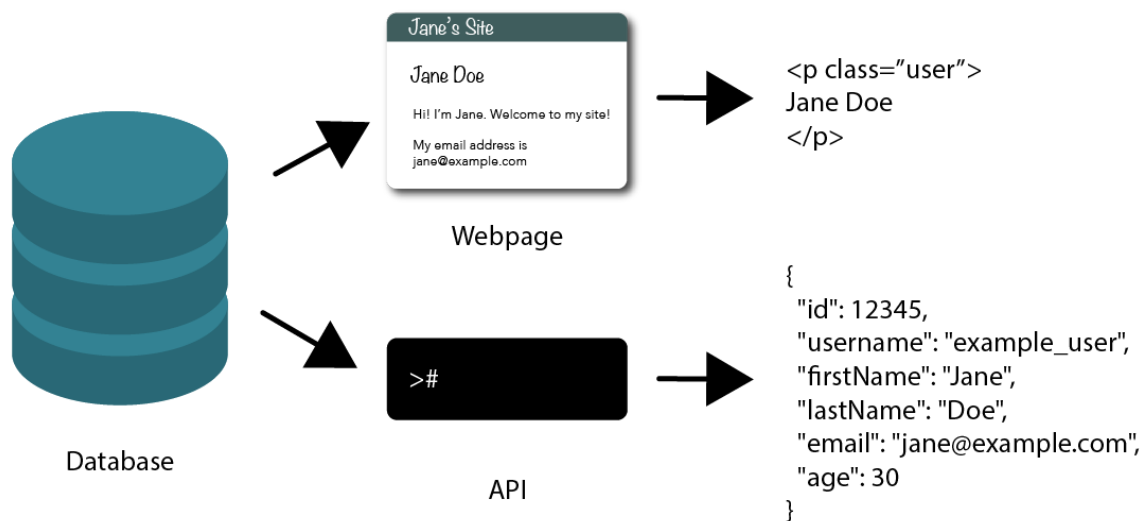
To access API information via the web, you can often use your web browser or JavaScript to open what is called an endpoint, which is simply a URL that returns structured information. For instance, EdTech Books has an open API that allows you to access content and other information via the following endpoint: https://edtechbooks.org/api.v2.php.

If you attempt to access this endpoint in your web browser, you will receive an error. This happens because the endpoint expects you to provide some information to help it know what data to return to you. In this case, try telling it that you want to search for books about technology by using this link: https://edtechbooks.org/api.v2.php?action=search&term=technology&entity_type=Book . In this link, we told the API to do the action "search," to look for the term "technology," and to return only "Book" entities. If you go to this URL, it will return any books on the site with the term "technology" in the title. Notice that the results are formatted in JavaScript Object Notation (JSON), which may be difficult for a human to read, but it is great for a computer.

To see the value of this approach, let's try accessing the same endpoint in the Inspector Console with JavaScript by running the following command:

```
fetch('https://edtechbooks.org/api.v2.php?action=search&term=technology&entity_type=Book')
  .then(response => response.json())
  .then(data => {
     for(var i in data.books) {
       console.log(data.books[i].name);
     }
  });
```

Fiddle with this a little to get results with different terms, like "research" or "chemistry," or check out the EdTech Books API documentation for more options.

Each API endpoint is different, meaning that it accepts different parameters and returns different types of results. So, they are great if you want to extract a lot of data from a single site, such as a repository, but they are less useful if you only want a little information from many sites.

Additionally, access to APIs is managed by the software developers, who determine what content can be accessed and by whom. Some sites allow open access to their APIs, some require a developer login or key, and others don't allow anyone outside of the organization to access them.

Some useful APIs that I have used in my work include the following:

- Twitter
- YouTube
- Scopus
- Google Custom Search
- EdTech Books

For most of these, you can have some level of free access, but you must create an account to request an API key to begin. You would then pass this key to the endpoint in your calls so that the API knows to give you access.

For instance, if you wanted to get a list of articles from the Scopus API for the journal *Educational Technology Research and Development* (ISSN 1556-6501), you could use the following endpoint after signing up and getting your own API key and instance token:

```
https://api.elsevier.com/content/search/scopus?query=issn(1556-6501)&apiKey=YOURAPIKEY&insttoken=YOURINSTTOKEN&count=25
```

Note that this returns an XML formatted document rather than JSON, but if you fetch this URL in your Inspector, you will get JSON, which you can then parse for titles or other information as follows:

```
fetch('https://api.elsevier.com/content/search/scopus?query=issn(1556-6501)&apiKey=YOURAPIKEY&insttoken=YOURINSTTOKEN&count=25')
  .then(response => response.json())
```

```
.then(data => {
    for(var i in data['search-results'].entry) {
        console.log(data['search-results'].entry[i]['dc:title']);
    }
});
```

This will return results like the following:

**Figure 2**

Example Results from a Scopus API Search

A Note on Security

If you must use a password or secret key to access an API in any way, then you should be aware that simply using JavaScript to pass these values to the API server (as in the Elsevier example above) will give others using your code unauthorized access. So, if you are sharing your code with others, be sure to remove keys or other authentication data to maintain security.

If you are developing an application that you would like to share with others that uses your credentials to access an API, then you will need to have a way to securely have your application communicate with the API server without revealing sensitive data (like keys) to your user. The typical way to do this is to have your own server that takes your user behaviors in your application and routes them to the API server on their behalf, overlaying or removing sensitive data as needed, as in Figure 3.

Figure 3

Example of an Application that Uses an API

Webpage or Application   Your Server   The API Server
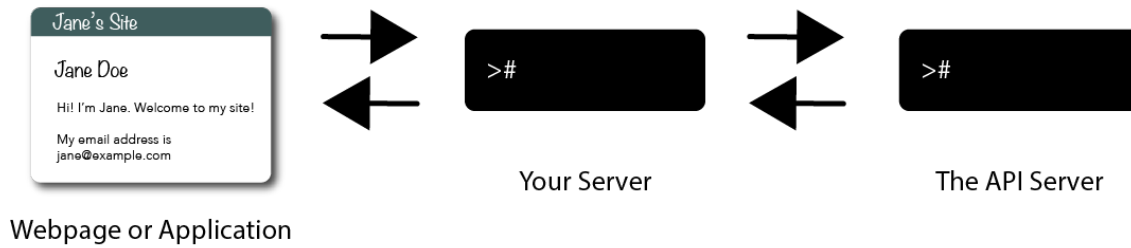
In this example, the API key would be stored on your server and sent to the API server with requests, but your end user on your webpage or application will never be able to see the API key or how your server communicates with the API server. This book will not guide you in setting up a server environment for hosting secure applications in this way, but I merely point this out here so that you will realize that any code you create or use in this book will need to be adapted for such uses and should not be shared with others in a way that includes compromising information (like security keys).