Web Development Pick-Me-Ups

# HTML

## Basic

* **HTML** stands for **H**yper**T**ext **M**arkup **L**anguage and is used to create the structure and content of a webpage.
* Most HTML elements contain opening and closing tags with raw text or other HTML tags between them.
* HTML elements can be nested inside other elements. The enclosed element is the child of the enclosing parent element.
* Any visible content should be placed within the opening and closing <body> tags.
* Headings and sub-headings, <h1> to <h6> tags, are used to enlarge text.
* <p>, <span> and <div> tags specify text or blocks.
* The <em> and <strong> tags are used to emphasize text.
* Line breaks are created with the <br> tag.
* Ordered lists (<ol>) are numbered and unordered lists (<ul>) are bulleted.
* Images (<img>) and videos (<video>) can be added by linking to an existing source.
* Syntax: <img src="smiley.gif" alt="Smiley face" height="42" width="42">
* “alt” is used to specify stuff when the image doesn’t load or give a description of the image
* Metadata is information about the page that isn’t displayed directly on the web page. Unlike the information inside of the <body> tag, the metadata in the head is information about the page itself.
* The <!DOCTYPE html> declaration should always be the first line of code in your HTML files. This lets the browser know what version of HTML to expect.
* The <html> element will contain all of your HTML code.
* Information about the web page, like the title, belongs within the <head> of the page.
* You can add a title to your web page by using the <title> element, inside of the head.
* A webpage’s title appears in a browser’s tab.
* Anchor tags (<a>) are used to link to internal pages, external pages, or content on the same page.
* You can create sections on a webpage and jump to them using <a> tags and adding ids to the elements you wish to jump to.
* Whitespace between HTML elements helps make code easier to read while not changing how elements appear in the browser.
* Indentation also helps make code easier to read. It makes parent-child relationships visible.
* Comments are written in HTML using the following syntax: <!-- comment -->.

## Tables

* The <table> element creates a table.
* The <tr> element adds rows to a table.
* To add data to a row, you can use the <td> element.
* Table headings clarify the meaning of data. Headings are added with the <th> element.
* Table data can span columns using the colspan attribute.
* Table data can span rows using the rowspan attribute.
* Tables can be split into three main sections: a head, a body, and a footer.
* A table’s head is created with the <thead> element.
* A table’s body is created with the <tbody> element.
* A table’s footer is created with the <tfoot> element.

## Forms

* The purpose of a <form> is to allow users to input information and send it.
* The <form>‘s action attribute determines where the form’s information goes.
* The <form>‘s method attribute determines how the information is sent and processed.
* To add fields for users to input information we use the <input> element and set the type attribute to a field of our choosing:
  + Setting type to "text" creates a single row field for text input.
  + Setting type to "password" creates a single row field that censors text input.
  + Setting type to "number" creates a single row field for number input.
  + Setting type to "range" creates a slider to select from a range of numbers.
  + Setting type to "checkbox" creates a single checkbox which can be paired with other checkboxes.
  + Setting type to "radio" creates a radio button that can be paired with other radio buttons.
  + Setting type to "list" will pair the <input> with a <datalist> element if the id of both are the same.
  + Setting type to "submit" creates a submit button.
  + The “placeholder” attribute is displayed in the field while there is no input and disappears after the user interacts. It can be used instead of the label
* A <select> element is populated with <option> elements and renders a dropdown list selection.
* A <datalist> element is populated with <option> elements and works with an <input> to search through choices.
* A <textarea> element is a text input field that has a customizable area.
* When a <form> is submitted, the name of the fields that accept input and the value of those fields are sent as name=value pairs.

### Validation

* Client-side validations happen in the browser before information is sent to a server.
* Adding the required attribute to an input related element will validate that the input field has information in it.
* Assigning a value to the min attribute of a number input element will validate an acceptable minimum value.
* Assigning a value to the max attribute of a number input element will validate an acceptable maximum value.
* Assigning a value to the minlength attribute of a text input element will validate an acceptable minimum number of characters.
* Assigning a value to the maxlength attribute of a text input element will validate an acceptable maximum number of characters.
* Assigning a regex to pattern matches the input to the provided regex.
* If validations on a <form> do not pass, the user gets a message explaining why and the <form> cannot be submitted.

## Semantic HTML

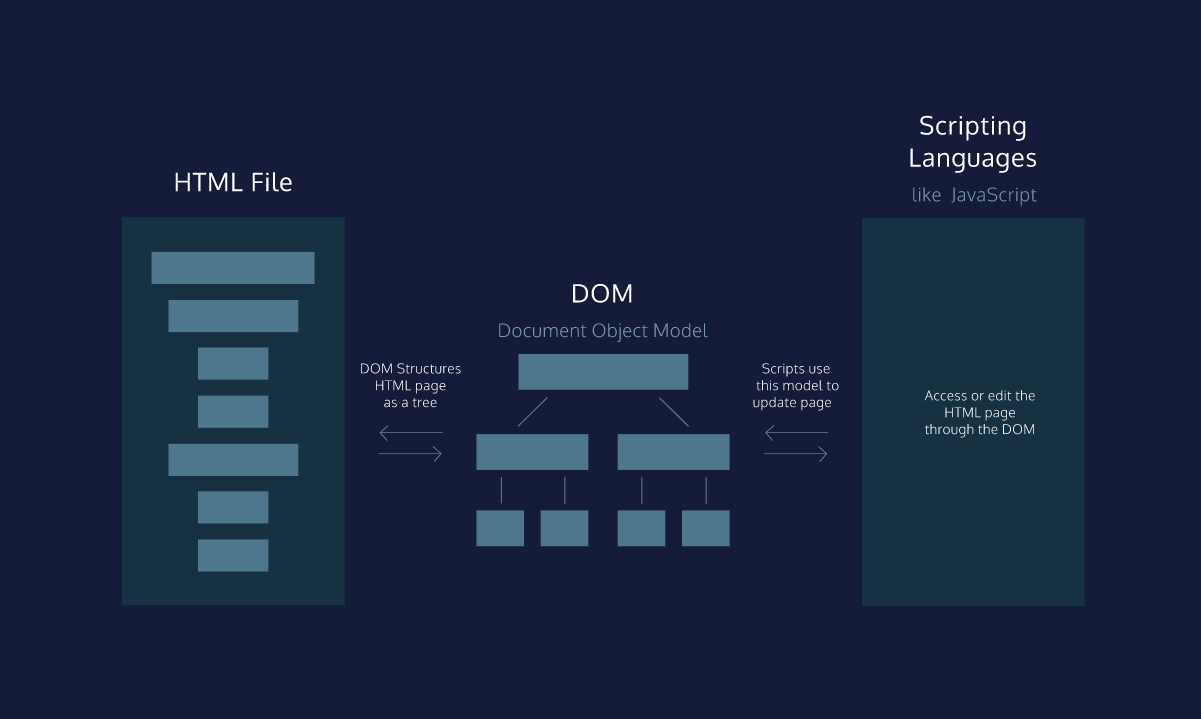
* Semantic HTML introduces meaning to a page through specific elements that provide context as to what is in between the tags.
* Semantic HTML is a modern standard and makes a website accessible for people who use screen readers to translate the webpage and improves your website’s SEO.
* <header>, <nav> , <main> and <footer> create the basic structure of the webpage.
* <section> defines elements in a document, such as chapters, headings, or any other area of the document with the same theme.
* <article> holds content that makes sense on its own such as articles, blogs, comments, etc.
* <aside> contains information that is related to the main content, but not required in order to understand the dominant information.
* <figure> encapsulates all types of media.
* <figcaption> is used to describe the media in <figure>.
* <video>, <embed>, and <audio> elements are used for media files.

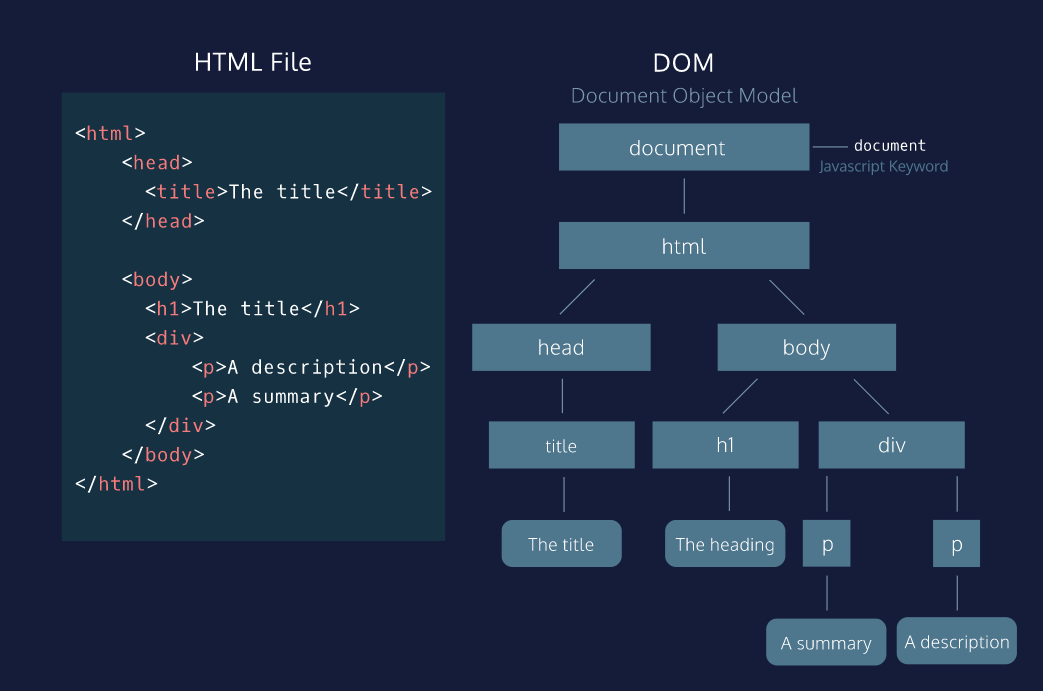
## Scripts

* HTML creates the skeleton of a webpage, but JavaScript introduces interactivity
* The <script> element has an opening and closing tag. You can embed JavaScript code inbetween the opening and closing <script> tags.
* You link to external JavaScript files with the **src** attribute in the opening <script> tag.
* By default, scripts are loaded and executed as soon as the HTML parser encounters them in the HTML file, the HTML parser waits to load the entire script before from proceeding to parse the rest of the page elements.
* The defer attribute ensures that the entire HTML file has been parsed before the script is executed.
* The async attribute will allow the HTML parser to continue parsing as the script is being downloaded, but will execute immediately after it has been downloaded.

The old convention was to put scripts right before the </body> tag to prevent the script from blocking the rest of the HTML content. Now, the convention is to put the script tag in the <head> element and to use the defer and async attributes.

## DOM





* The DOM is a structural model of a web page that allows for scripting languages to access that page.
* The system of organization in the DOM mimics the nesting structure of an HTML document.
* Elements nested within another are referred to as the children of that element. The element they are nested within is called the parent element of those elements.
* The DOM also allows access to the regular attributes of an HTML element such as its style, id, etc.
* The document keyword grants access to the root of the DOM in JavaScript
* The DOM Interface allows you to select a specific element with CSS selectors by using the .querySelector() method
* You can also access an element directly by its ID with .getElementById()
* The .innerHTML and .style properties allow you to modify an element by changing its contents or style respectively
* You can create, append, and remove elements by using the .createElement(),.appendChild() and .removeChild() methods respectively
* The .onclick property can add interactivity to a DOM element based on a click event
* For example, the following code selects the first element with a class of blue and assigns blue as the background-color:

let blueElement = document.querySelector('.blue');

blueElement.style.backgroundColor = 'blue';

## Handlebars

* Handlebars is an external library used to create templates which are a mix of HTML, text, and expressions.
* Handlebars uses expressions which are wrapped inside double braces like: {{someVariable}}
* A script tag with a type of "text/x-handlebars-template" is used to deliver a template to the browser.
* Handlebar.compile() returns a templating function from a template script intended for Handlebars.
* A template created from .compile() will take an object as an argument and use it as context to generate a string containing HTML.
* Handlebars has built in block helpers which can be included in a Handlebars script.
* Block helpers have a starting expression and an ending expression. The starting expression will have a # appears before a keyword. The ending expression will have the same keyword but with a / character to denote the end.
* The {{if}} will conditionally render a block of code.
* An {{else}} expression can be inserted into an if helper block and used as part of the conditional statement.
* {{each}} is another built-in helper expression which accepts an an array to iterate through.
* In the block helper functions, the {{this}} expression gives context and serves as a placeholder for the current value.

## Requests

* **HTTP & TCP: How it Works**

When you type an address such as [www.codecademy.com](https://codecademy.com/) into your browser, you are commanding it to open a TCP channel to the server that responds to that URL (or Uniform Resource Locator, which you can read more about on [Wikipedia](https://en.wikipedia.org/wiki/Uniform_Resource_Locator)). A URL is like your home address or phone number because it describes how to reach you.

In this situation, your computer, which is making the request, is called the client. The URL you are requesting is the address that belongs to the server.

Once the TCP connection is established, the client sends a HTTP GET request to the server to retrieve the webpage it should display. After the server has sent the response, it closes the TCP connection. If you open the website in your browser again, or if your browser automatically requests something from the server, a new connection is opened which follows the same process described above.

* JavaScript is the language of the web because of its asynchronous capabilities. AJAX, which stands for Asynchronous JavaScript and XML, is a set of tools that are used together to take advantage of JavaScript’s asynchronous capabilities.
* There are many HTTP request methods, two of which are GET and POST.
* GET requests only request information from other sources.
* POST methods can introduce new information to other sources in addition to requesting it.
* GET requests can be written using an XMLHttpRequest object and vanilla JavaScript.
* POST requests can also be written using an XMLHttpRequest object and vanilla JavaScript.
* Writing GET and POST requests with XHR objects and vanilla JavaScript requires constructing the XHR object using new, setting the responseType, creating a function that will handle the response object, and opening and sending the request.
* To add a query string to a URL endpoint you can use ? and include a parameter.
* To provide additional parameters, use & and then include a key-value pair, joined by =.
* Determining how to correctly write the requests and how to properly implement them requires carefully reading the documentation of the API with which you’re working.

### Standard HTTP:

const https = require('https');

https.get('https://api.nasa.gov/planetary/apod?api\_key=DEMO\_KEY', (resp) => {

let data = '';

// A chunk of data has been recieved.

resp.on('data', (chunk) => {

data += chunk;

});

// The whole response has been received. Print out the result.

resp.on('end', () => {

console.log(JSON.parse(data).explanation);

});

}).on("error", (err) => {

console.log("Error: " + err.message);

});

### HTTPS using async and promises:

'use strict';

const fs = require('fs');

const https = require('https');

process.stdin.resume();

process.stdin.setEncoding('utf-8');

let inputString = '';

let currentLine = 0;

process.stdin.on('data', function(inputStdin) {

    inputString += inputStdin;

});

process.stdin.on('end', function() {

    inputString = inputString.split('\n');

    main();

});

function readLine() {

    return inputString[currentLine++];

}

async function getCountryName(code) {

    // API endpoint: https://jsonmock.hackerrank.com/api/countries?page=<PAGE\_NUMBER>

    let answer='';

   for(let n=1;n<26;n++){

    const data=await doRequest(n);

    console.log(data);

    answer=search(data.data,code);

    if(answer!='') return answer;

   }

   return '';

}

function doRequest(page){

    return new Promise((resolve, reject)=>{

    const url=`https://jsonmock.hackerrank.com/api/countries?page=${page}`;

    const req=https.get(url, (resp)=>{

        let data='';

        resp.on('data',(chunk)=>{

            data+=chunk;

        });

        resp.on('end',()=>{

            resolve(JSON.parse(data));

        });

    });

    req.on("error",(err)=>{

        console.log("Error: "+err.message);

    });

    req.end();

    });

}

function search(arr, code){

    for(let i=0; i< arr.length; i++){

        if(arr[i].alpha2Code==code){

            return arr[i].name;

        }

    }

    return '';

}

async function main() {

//read input to code

console.log(await getCountryName(code));

}

### XHR GET:



* Query String(?): A query string contains additional information to be sent with a request. A query string is separated from the URL using a ? character. After ?, you can then create a parameter which is a key value pair joined by a =. Examine the example below:

'https://api.datamuse.com/words?key=value'

If you want to add an additional parameter you will have to use the & character to separate your parameters. Like so:

'https://api.datamuse.com/words?key=value&anotherKey=anotherValue'

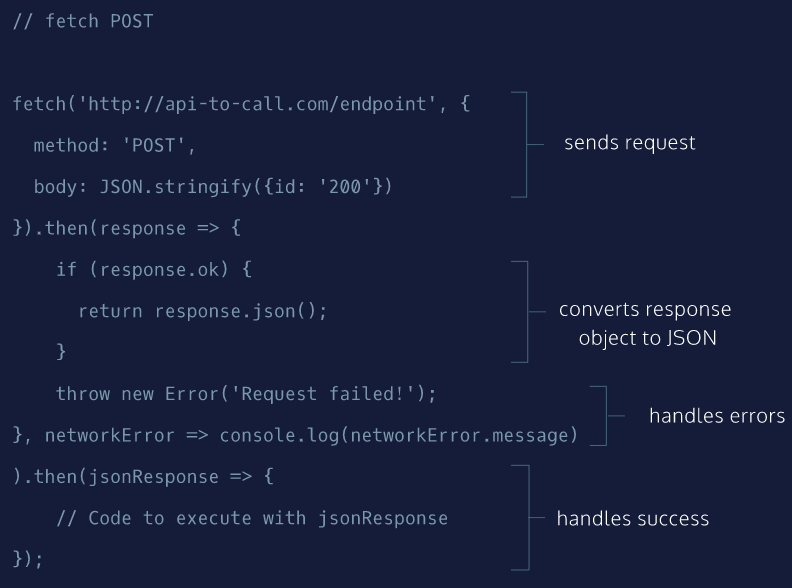
### XHR POST:



### Fetch API:

* GET and POST requests can be created a variety of ways.
* Use AJAX to asynchronously request data from APIs. fetch() and async/await are new functionalities developed in ES6 (promises) and ES8 respectively.
* Promises are a new type of JavaScript object that represent data that will eventually be returned from a request.
* fetch() is a web API that can be used to create requests. fetch() will return promises.
* We can chain .then() methods to handle promises returned by fetch().
* The .json() method converts a returned promise to a JSON object.
* async is a keyword that is used to create functions that will return promises.
* await is a keyword that is used to tell a program to continue moving through the message queue while a promise resolves.
* await can only be used within functions declared with async.





### Fetch with Async:





# CSS

CSS, or **Cascading Style Sheets**, is a language that web developers use to style the HTML content on a web page. If you’re interested in modifying colors, font types, font sizes, shadows, images, element positioning, and more, CSS is the tool for the job!

## Intro

* CSS can change the look of HTML elements. In order to do this, CSS must select HTML elements, then apply styles to them.
* CSS can select HTML elements by tag, class, or ID.
* Multiple CSS classes can be applied to one HTML element.
* Classes can be reusable, while IDs can only be used once.
* IDs are more specific than classes, and classes are more specific than tags. That means IDs will override any styles from a class, and classes will override any styles from a tag selector.
* Multiple selectors can be chained together to select an element. This raises the specificity, but can be necessary.
* Eg: { .nutrition ui }– means ui child inside nutrition class
* { ui.nutrition} – means ui tag *with* class nutrition
* Nested elements can be selected by separating selectors with a space.
* The !important flag will override any style, however it should almost never be used, as it is extremely difficult to override.
* Multiple unrelated selectors can receive the same styles by separating the selector names with commas.

## Rule-Sets

* CSS declarations are structured into property and value pairs.
* The font-family property defines the typeface of an element.
* font-size controls the size of text displayed.
* font-weight defines how thin or thick text is displayed.
* The text-align property places text in the left, right, or center of its parent container.
* Text can have two different color attributes: color and background-color. color defines the color of the text, while background-color defines the color behind the text.
* CSS can make an element transparent with the opacity property.
* CSS can also set the background of an element to an image with the background-image property. Syntax: background-image: url(“location-of-image.png”);
* A page’s styling (font sizes, colors, etc.) is outlined in a design spec, which is a standard document you’d expect to receive as a freelance web developer.

Eg:

## Data Paths

* Form: “D:/Folder/AnotherFolder/file.typ”
* Notations: “./” used to denote the directory of the current file you’re working on.

Eg. For index.html, the folder which contains it is denoted as “./”

* Notation: “../” used to denote the directory containing the current files directory (if the current is not the root).

Eg. If path to file is “D:/folder/root/sub-root/file.html”, then “../” denotes: “D:/folder/root/”

And “./” denotes “D:/folder/root/sub-root”

## The Box Model

1. The box model comprises a set of properties used to create space around and between HTML elements.
2. The height and width of a content area can be set in pixels or percentage.
3. Borders surround the content area and padding of an element. The color, style, and thickness of a border can be set with CSS properties.
4. Padding is the space between the content area and the border. It can be set in pixels or percent.
5. Margin is the amount of spacing outside of an element’s border.
6. Horizontal margins add, so the total space between the borders of adjacent elements is equal to the sum of the right margin of one element and the left margin of the adjacent element.
7. Vertical margins collapse, so the space between vertically adjacent elements is equal to the larger margin.
8. margin: 0 auto horizontally centers an element inside of its parent content area, if it has a width.
9. The overflow property can be set to display, hide, or scroll, and dictates how HTML will render content that overflows its parent’s content area.
10. The visibility property can hide or show elements.

Box Model: Content-box (box-sizing: content-box)

Box Model: Border-Box

1. In the default box model, box dimensions are affected by border thickness and padding.
2. The box-sizing property controls the box model used by the browser.
3. The default value of the box-sizing property is content-box.
4. The value for the new box model is border-box.
5. The border-box model is not affected by border thickness or padding.
6. border-radius: controls curvature at the corners

* In use <class/element/tag name>:hover to make it change during hover, and have 2 properties, eg:
  + .class{
  + //Default look
  + }
  + .class:hover{
  + //hover look
  + }

## Positioning

* A browser will render the elements of an HTML document that has no CSS from left to right, top to bottom, in the same order as they exist in the document. This is called the flow of elements in HTML.
* The position property allows you to specify the position of an element in three different ways.
* When set to relative, an element’s position is relative to its default position (not the PARENTS position) on the page.
* When set to absolute, an element’s position is relative to its closest positioned parent element (or nearest non-static element). It can be pinned to any part of the web page, but the element will still move with the rest of the document when the page is scrolled.
* When set to fixed, an element’s position can be pinned to any part of the web page. The element will remain in view no matter what.
* The z-index of an element specifies how far back or how far forward an element appears on the page when it overlaps other elements. Ignored by static elements
* The display property allows you control how an element flows vertically and horizontally a document.
* inline elements take up as little space as possible, and they cannot have manually-adjusted width or height.
* block elements take up the width of their container and can have manually-adjusted heights.
* inline-block elements can have set width and height, but they can also appear next to each other and do not take up their entire container width. Doesn’t start new lines, can flow horizontally with siblings
* The float property can move elements as far left or as far right as possible on a web page.
* You can clear an element’s left or right side (or both) using the clear property.

### Responsive Design

* Content on a website can be sized relative to other elements on the page using *relative measurements*.
* The unit of em sizes font relative to the font size of a parent element.
* The unit of rem sizes font relative to the font size of a root element. That root element is the <html> element.
* Percentages are commonly used to size box-model features, like the width, height, padding, or margin of an element.
* When percentages are used to size width and height, child elements will be sized relative to the dimensions of their parent (remember that parent dimensions must first be set).
* Percentages can be used to set padding and margin. Horizontal and vertical padding and margin are set relative to the width of a parent element.
* The minimum and maximum width of elements can be set using min-width and max-width.
* The minimum and maximum height of elements can be set using min-height and max-height.
* When the height of an image or video is set, then its width can be set to auto so that the media scales proportionally. Reversing these two properties and values will also achieve the same result.
* A background image of an HTML element will scale proportionally when its background-size property is set to cover.
* A very common pattern to size images properly:

.container {

width: 50%;

height: 200px;

overflow: hidden;

}

.container img {

display: block;

max-width: 100%;

height: auto;

or

max-height: 100%;

width: auto

}



* When a website responds to the size of the screen it’s viewed on, it’s called a *responsive* website.
* You can write *media queries* to help with different screen sizes.
* Media queries require *media features*. Media features are the conditions that must be met to render the CSS within a media query.
* Media features can detect many aspects of a user’s browser, including the screen’s width, height, resolution, orientation, and more.
* The and operator requires multiple media features to be true at once.
* A comma separated list of media features only requires one media feature to be true for the code within to be applied.
* The best practice for identifying where media queries should be set is by resizing the browser to determine where the content naturally breaks. Natural breakpoints are found by resizing the browser.

### Flexbox

1. display: flex changes an element to a block-level container with flex items inside of it.
2. display: inline-flex allows multiple flex containers to appear inline with each other.
3. justify-content is used to space items along the major axis.
4. align-items is used to space items along the cross axis.
5. flex-grow is used to specify how much space (and in what proportions) flex items absorb along the major axis.
6. flex-shrink is used to specify how much flex items shrink and in what proportions along the major axis.
7. flex-basis is used to specify the initial size of an element styled with flex-grow and/or flex-shrink.
8. flex is used to specify flex-grow, flex-shrink, and flex-basis in one declaration.
9. flex-wrap specifies that elements should shift along the cross axis if the flex container is not large enough.
10. align-content is used to space rows along the cross axis.
11. flex-direction is used to specify the major and cross axes.
12. flex-flow is used to specify flex-wrap and flex-direction in one declaration.
13. Flex containers can be nested inside of each other by declaring display: flex or display: inline-flex for children of flex containers.
    1. There are five values for the justify-content property:
14. flex-start — all items will be positioned in order starting, from the left of the parent container, with no extra space between or before them.
15. flex-end — all items will be positioned in order, with the last item starting on the right side of the parent container, with no extra space between or after them.
16. center — all items will be positioned in order, in the center of the parent container with no extra space before, between, or after them.
17. space-around — items will be positioned with equal space before and after each item, resulting in double the space between elements.
18. space-between — items will be positioned with equal space between them, but no extra space before the first or after the last elements.
    1. There are five values we can use for the align-items property:
19. flex-start — all elements will be positioned at the top of the parent container.
20. flex-end — all elements will be positioned at the bottom of the parent container.
21. center — the center of all elements will be positioned halfway between the top and bottom of the parent container.
22. baseline — the bottom of the content of all items will be aligned with each other.
23. stretch — if possible, the items will stretch from top to bottom of the container (this is the default value; elements with a specified height will not stretch; elements with a minimum height or no height specified will stretch).
    1. align-content accepts six values:
24. flex-start — all rows of elements will be positioned at the top of the parent container with no extra space between.
25. flex-end — all rows of elements will be positioned at the bottom of the parent container with no extra space between.
26. center — all rows of elements will be positioned at the center of the parent element with no extra space between.
27. space-between — all rows of elements will be spaced evenly from the top to the bottom of the container with no space above the first or below the last.
28. space-around — all rows of elements will be spaced evenly from the top to the bottom of the container with the same amount of space at the top and bottom and between each element.
29. stretch — if a minimum height or no height is specified, the rows of elements will stretch to fill the parent container from top to bottom (default value).
    1. The flex-direction property can accept four values:

* row — elements will be positioned from left to right across the parent element starting from the top left corner (default).
* row-reverse — elements will be positioned from right to left across the parent element starting from the top right corner.
* column — elements will be positioned from top to bottom of the parent element starting from the top left corner.
* column-reverse — elements will be positioned from the bottom to the top of the parent element starting from the bottom left corner.

Shortcut: **flex-flow: flex-direction flex-wrap;**

## Colors

There are four ways to represent color in CSS:

* Named colors — there are 147 named colors, which you can review [here](https://msdn.microsoft.com/en-us/library/aa358802(v=vs.85).aspx).
* Hexadecimal or hex colors
  + Hexadecimal is a number system with has sixteen digits, 0 to 9 followed by “A” to “F”.
  + Hex values always begin with # and specify values of red, blue and green using hexademical numbers such as #23F41A.
* RGB
  + RGB colors use the rgb() syntax with one value for red, one value for blue and one value for green.
  + RGB values range from 0 to 255 and look like this: rgb(7, 210, 50).
* HSL
  + HSL stands for hue (the color itself), saturation (the intensity of the color), and lightness (how light or dark a color is).
  + Hue ranges from 0 to 360 and saturation and lightness are both represented as percentages like this: hsl(200, 20%, 50%).
* You can add opacity to color in RGB and HSL by adding a fourth value, a, which is represented as a percentage.

## Typography

* Typography is the art of arranging text on a page.
* Text can appear in any number of weights, with the font-weight property.
* Text can appear in italics with the font-style property.
* The vertical spacing between lines of text can be modified with the line-height property.
* Serif fonts have extra details on the ends of each letter. Sans-Serif fonts do not.
* Fallback fonts are used when a certain font is not installed on a user’s computer.
* Google Fonts provides free fonts that can be used in an HTML file with the <link> tag or the @font-face property.
* Local fonts can be added to a document with the @font-face property and the path to the font’s source.
* The word-spacing property changes how far apart individual words are.
* The letter-spacing property changes how far apart individual letters are.
* The text-align property changes the horizontal alignment of text.
* Line Spacing



* Serif and Sans Serif:





# JavaScript

JavaScript is a powerful, flexible, and fast programming language now being used for increasingly complex web development and beyond!

## Intro

* Data is printed, or logged, to the console, a panel that displays messages, with console.log().
* We can write single-line comments with // and multi-line comments between /\* and \*/.
* There are 7 fundamental data types in JavaScript: strings, numbers, booleans, null, undefined, symbol, and object.
* Numbers are any number without quotes: 23.8879
* Strings are characters wrapped in single or double quotes: 'Sample String'
* The built-in arithmetic operators include +, -, \*, /, and %.
* Objects, including instances of data types, can have properties, stored information. The properties are denoted with a . after the name of the object, for example: 'Hello'.length.
* Objects, including instances of data types, can have methods which perform actions. Methods are called by appending the object or instance with a period, the method name, and parentheses. For example: 'hello'.toUpperCase().
* We can access properties and methods by using the ., dot operator.
* Built-in objects, including Math, are collections of methods and properties that JavaScript provides.

## Variables

* Variables hold reusable data in a program and associate it with a name.
* Variables are stored in memory.
* The var keyword is used in pre-ES6 versions of JS.
* let is the preferred way to declare a variable when it can be reassigned, and const is the preferred way to declare a variable with a constant value.
* Variables that have not been initialized store the primitive data type undefined.
* Mathematical assignment operators make it easy to calculate a new value and assign it to the same variable.
* The + operator is used to concatenate strings including string values held in variables
* In ES6, template literals use backticks ` and ${} to interpolate values into a string.
* The typeof keyword returns the data type (as a string) of a value.

## Conditionals

* An if statement checks a condition and will execute a task if that condition evaluates to true.
* if...else statements make binary decisions and execute different code blocks based on a provided condition.
* We can add more conditions using else if statements.
* Comparison operators, including <, >, <=, >=, ===, and !== can compare two values.
* The logical and operator, &&, or “and”, checks if both provided expressions are truthy.
* The logical operator ||, or “or”, checks if either provided expression is truthy.
* The bang operator, !, switches the truthiness and falsiness of a value.
* The ternary operator is shorthand to simplify concise if...else statements.
* A switch statement can be used to simplify the process of writing multiple else if statements. The break keyword stops the remaining cases from being checked and executed in a switch statement.

The list of falsy values includes:

* 0
* Empty strings like "" or ''
* null which represent when there is no value at all
* undefined which represent when a declared variable lacks a value
* NaN, or Not a Number

### Short-circuit evaluation:

Let name= username || ‘stranger’;

Here, if value of username is falsy, ‘stranger’ gets assigned to ‘name’

## Functions



* A function is a reusable block of code that groups together a sequence of statements to perform a specific task.
* A parameter is a named variable inside a function’s block which will be assigned the value of the argument passed in when the function is invoked:



* ES6 introduces new ways of handling arbitrary parameters through default parameters which allow us to assign a default value to a parameter in case no argument is passed into the function.
* To return a value from a function, we use a return statement.
* To define a function using function expressions:
* Function definition can be made concise using concise arrow notation:

## Scope

* **Scope** is the idea in programming that some variables are accessible/inaccessible from other parts of the program.
* **Blocks** are statements that exist within curly braces {}.
* **Global scope** refers to the context within which variables are accessible to every part of the program.
* **Global variables** are variables that exist within global scope.
* **Block scope** refers to the context within which variables that are accessible only within the block they are defined.
* **Local variables** are variables that exist within block scope.
* **Global namespace** is the space in our code that contains globally scoped information.
* **Scope pollution** is when too many variables exist in a namespace or variable names are reused.

## Arrays

* Arrays are lists that store data in JavaScript.
* Arrays are created with brackets [].
* Each item inside of an array is at a numbered position, or index, starting at 0.
* We can access one item in an array using its index, with syntax like: myArray[0].
* We can also change an item in an array using its index, with syntax like myArray[0] = 'new string';
* Arrays have a length property, which allows you to see how many items are in an array.
* Arrays have their own methods, including .push() and .pop(), which add and remove items from an array, respectively.
* Arrays have many methods that perform different tasks, such as .slice() and .shift(), you can find documentation at the [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) website.
* Some built-in methods are mutating, meaning the method will change the array, while others are not mutating. You can always check the documentation.
* Variables that contain arrays can be declared with let or const. Even when declared with const, arrays are still mutable. However, a variable declared with const cannot be reassigned.
* Arrays mutated inside of a function will keep that change even outside the function.
* Arrays can be nested inside other arrays.
* To access elements in nested arrays chain indices using bracket notation.
* .splice() like this:

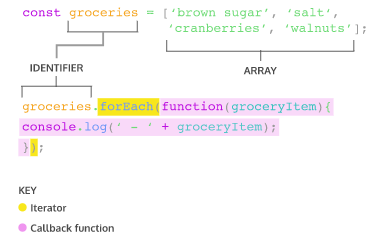
array.splice(indexToStart, numberOfIndices, 'stringToAdd');

* .join() like this: array.

## Loops

* Loops perform repetitive actions so we don’t have to code that process manually every time.
* How to write for loops with an iterator variable that increments or decrements
* How to use a for loop to iterate through an array
* A nested for loop is a loop inside another loop
* while loops allow for different types of stopping conditions
* Stopping conditions are crucial for avoiding infinite loops.
* do...while loops run code at least once— only checking the stopping condition after the first execution
* The break keyword allows programs to leave a loop during the execution of its block

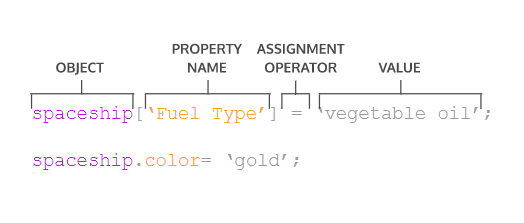
### Iterators

* .forEach() is used to execute the same code on every element in an array but does not change the array and returns undefined. 
* .map() executes the same code on every element in an array and returns a new array with the updated elements.
* .filter() checks every element in an array to see if it meets certain criteria and returns a new array with the elements that return truthy for the criteria.
* .findIndex() returns the index of the first element of an array which satisfies a condition in the callback function. It returns -1 if none of the elements in the array satisfies the condition.
* .reduce() iterates through an array and takes the values of the elements and returns a single value.
* .some() iterates through an array and returns true if there is atleast one callback returning a truthy value
* .every() iterates through an array and returns true if all members return truthy on the calback function
* All iterator methods takes a callback function that can be pre-defined, or a function expression, or an arrow function.

## Higher-Order Functions

* Abstraction allows us to write complicated code in a way that’s easy to reuse, debug, and understand for human readers
* We can work with functions the same way we would any other type of data including reassigning them to new variables
* JavaScript functions are first-class objects, so they have properties and methods like any object
* Functions can be passed into other functions as parameters
* A higher-order function is a function that either accepts functions as parameters, returns a function, or both

## Objects

* Objects store collections of key-value pairs.
* Each key-value pair is a property—when a property is a function it is known as a method.
* An object literal is composed of comma-separated key-value pairs surrounded by curly braces.
* You can access, add or edit a property within an object by using dot notation or bracket notation.
* We can add methods to our object literals using key-value syntax with anonymous function expressions as values or by using the new ES6 method syntax.
* We can navigate complex, nested objects by chaining operators.
* Objects are mutable—we can change their properties even when they’re declared with const.
* Objects are passed by reference— when we make changes to an object passed into a function, those changes are permanent.
* We can iterate through objects using the For...in syntax.
  + For(let key in object){

Console.log(key); //prints name of key

Console.log(object[key]); //prints value

}

delete spaceship.mission; // Removes the mission property

DON’T FORGET **COMMA** BETWEEN Functions or key-value pairs

* The object that a method belongs to is called the calling object.
* The this keyword refers the calling object and can be used to access properties of the calling object.
* Methods do not automatically have access to other internal properties of the calling object.
* The value of this depends on where the this is being accessed from.
* We cannot use arrow functions as methods if we want to access other internal properties.
* JavaScript objects do not have built-in privacy, rather there are conventions to follow to notify other developers about the intent of the code.
* The usage of an underscore before a property name means that the original developer did not intend for that property to be directly changed.
* Setters and getter methods allow for more detailed ways of accessing and assigning properties.
* Factory functions allow us to create object instances quickly and repeatedly.  
  eg. Const factFunc =(key1) => {return { key1: key1, key2: key1, key3: ‘yo’,} };
* There are different ways to use object destructuring: one way is the property value shorthand and another is destructured assignment.
* Property value shorthand:
  + Const factFunc =(key1, key2) => {return { key1, key2};
* Destructures assignment:
  + const key1 = oldObject.key1;
  + Is same as: const { key1 } = oldObject;
* Built in methods:
  + Object.assign(target, source): adds source to target and returns target, this MODIFIES target
  + Object.keys(object): returns array of all keys in object
  + Object.entries(object): returns array of arrays of key-value pairs
* Tip: const obj = (name){

\_name,

Get name(){

return this.\_name; /\*error, cuz ‘\_’ before property, but not in parameter\*/

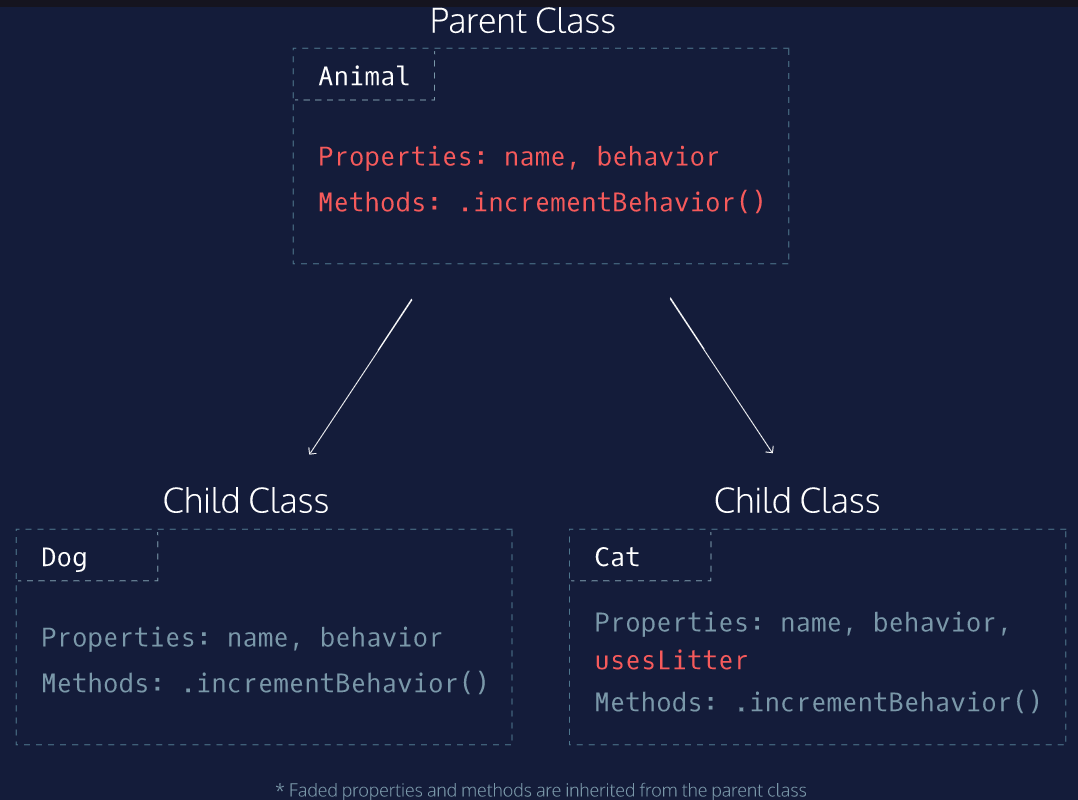
},

}

## Events

* JavaScript engines register events as objects with properties and methods associated with them.
* Event handlers are registered as properties of their event object.
* Event object properties like .target, .type, and .timeStamp are used to provide information about the event.
* eventTarget.addEventListener('click', eventHandlerFunction);
* The .addEventListener() method can be used to add multiple event handler functions to a single event.
* The .removeEventListener() method stops specific event handlers from “listening” for specific events firing.
* Eg. eventTarget.addEventListener('click', eventHandlerFunction);

## Classes

* Classes are templates for objects.
* Javascript calls a constructor method when we create a new instance of a class.
* Inheritance is when we create a parent class with properties and methods that we can extend to child classes. 
* We use the extends keyword to create a subclass.
* The super keyword calls the constructor() of a parent class.
* Static methods are called on the class, but not on instances of the class.
* Class method and getter syntax is the same as it is for objects **except you can not include commas between methods**.

## Modules

Modules in Node.js are reusable pieces of code that can be exported from one program and imported for use in another program.

* module.exports exports the module for use in another program.
* require() imports the module for use in the current program. require() is a JavaScript function that loads a module. It’s argument is the file path of the module: ./menu.js. With require(), the .js extension is optional and will be assumed if it is not included.

ES6 introduced a more flexible, easier syntax to export modules:

* default exports use export default to export JavaScript objects, functions, and primitive data types.
* named exports use the export keyword to export data in variables.
* named exports can be aliased with the as keyword.
* import is a keyword that imports any object, function, or data type.

## Debugging Tips

1. **Is your code throwing errors?** If so, read the error stack trace for the type, description, and location of the error. Go to the error’s location and try to fix.
2. **Is your code broken but not throwing errors?** Walk through your code using console.log() statements. When unexpected results occur, isolate the bug and try to fix it.
3. **Did you locate the bug using steps 1 and 2, but can’t fix the bug?** Consult documentation to make sure you are using all JavaScript functionality properly. If you are still stuck, Google your issue and consult Stack Overflow for help. Read solutions or post your own Stack Overflow question if none exist on the topic.

## Error Handling

* Errors will prevent a program from executing unless it is handled.

try {

throw Error('This error will get caught');

} catch (e) {

console.log(e); }

## Promises

Promises are objects that represent the eventual outcome of an asynchronous operation. A Promise object can be in one of three states:

* **Pending**: The initial state— the operation has not completed yet.
* **Fulfilled**: The operation has completed successfully and the promise now has a *resolved value*. For example, a request’s promise might resolve with a JSON object as its value.
* **Rejected**: The operation has failed and the promise has a reason for the failure. This reason is usually an Error of some kind.

We refer to a promise as *settled* if it is no longer pending— it is either fulfilled or rejected.

const executorFunction = (resolve, reject) => {

if (someCondition) {

resolve('I resolved!');

} else {

reject('I rejected!');

}

}

const myFirstPromise = new Promise(executorFunction);

* Promises are JavaScript objects that represent the eventual result of an asynchronous operation.
* Promises can be in one of three states: pending, resolved, or rejected.
* A promise is settled if it is either resolved or rejected.
* We construct a promise by using the new keyword and passing an executor function to the Promise constructor method.
* setTimeout() is a Node function which delays the execution of a callback function using the event-loop.
* We use .then() with a success handler callback containing the logic for what should happen if a promise resolves.
* We use .catch() with a failure handler callback containing the logic for what should happen if a promise rejects.
* Promise composition enables us to write complex, asynchronous code that’s still readable. We do this by chaining multiple .then()‘s and .catch()‘s.
* To use promise composition correctly, we have to remember to return promises constructed within a .then().
* We should chain multiple promises rather than nesting them.
* To take advantage of concurrency, we can use Promise.all().

## Async Functions

async functions always return a promise. This means we can use traditional promise syntax, like .then() and .catch with our async functions. An async function will return in one of three ways:

* If there’s nothing returned from the function, it will return a promise with a resolved value of undefined.
* If there’s a non-promise value returned from the function, it will return a promise resolved to that value.
* If a promise is returned from the function, it will simply return that promise
* async...await is syntactic sugar built on native JavaScript promises and generators.
* We declare an async function with the keyword async.
* Inside an async function we use the await operator to pause execution of our function until an asynchronous action completes and the awaited promise is no longer pending .
* await returns the resolved value of the awaited promise.
* We can write multiple await statements to produce code that reads like synchronous code.
* We use try...catch statements within our async functions for error handling.
* We should still take advantage of concurrency by writing async functions that allow asynchronous actions to happen in concurrently whenever possible.

*await* keyword returns the resolved value of the promise:

eg: let val=new Promise(resolved,reject) OP: Promise: pending

let val=await new Promise(resolved, reject) OP: resolved

using error handling with async:

async function hostDinnerParty(){

  try{

    let resolved=await myPromise();

    console.log(`${resolved}!`);

  } catch(error){ //Prints error

    console.log(error);

    console.log('Ordering a pizza!');

  };

};

* if we have multiple truly independent promises that we would like to execute fully in parallel, we must use individual .then() functions and avoid halting our execution with await.

# Abbreviations

|  |  |
| --- | --- |
| Short | Full |
| **AJAX**  API  **APK**  BASH  **CLI**  CSS  **DNS**  grep  **HTML**  HTTP  **IT**  JPG/JPEG  **JSON**  PHP  **REST**  **sed**  SEO  **SHA**  **SIM**  SQL  **URL**  VIRUS  **XHR**  XML  **XMPP** | **Asynchronous Javascript And XML**  Application Programming Interface  **Android application PacKage**  Bourne-Again SHell  **Command Line Interfaces**  Cascading Style Sheets  **Domain Name System**  Global Regular Expression Print  **Hyper Text Markup Language**  HyperText Transfer Protocol  **Information Technology**  Joint Photographic Experts Group  **JavaScript Object Notation**  HyperText Preprocessor  (Personal HomePage)  **REpresentational State Transfer**  **Stream Editor**  Search Engine Optimization  **Secure Hash Algorithm**  **Subscriber Identity Module**  Structured Query Language  **Uniform Resource Locator**  Vital Information Resource Under Seige  **XML HTTP Request**  eXtensible Markup Language  **eXtensible Messaging Presence Protocol** |