HTML

<html>

all code goes in here

</html>

<head> contains the Metadata which is the info about the page that isn’t displayed. This is the information about the code itself

Anything between the <html> will be interpreted as html code

<!DOCTYPE html> - this must be your first line of code when using html

There are 6 different heading <h1> </h1>  to <h6> </h6> - the h1 being the lasrges

<div> </div> is the division or a section in an html document.

Attributes are made up of 2 parts – The name and value of attribute

One commonly used attribute is the id. We can use the id attribute to specify different content (such as <div>s)  <div id="introduction">

* *Paragraphs* (<p>) contain a block of plain text.
* <span> contains short pieces of text or other HTML. They are used to separate small pieces of content that are on the same line as other content.
* The <em> tag will generally render as *italic* emphasis.
* The <strong> will generally render as **bold** emphasis.
* The spacing between code in an HTML file doesn't affect the positioning of elements in the browser. If you are interested in modifying the spacing in the browser, you can use HTML's *line break* element: <br>.
* <ul> - is an unordered list –
  + <ul> <li> </li> </ul> (the li add bullet points
* <ol> are like unordered except that they are numbered
  + <ol> <li></li> </ol>

Image = <img src="URL" /> This will add an image

<img src="#" alt="A field of yellow sunflowers" />  - the alt= feature is used for alternate text, it brings meaning to the images on our site

<video></video> is used to add video, must have opening/closing tag

<video src="myVideo.mp4" width="320" height="240" controls> Video not supported </video>

After the src attribute, the width and height attributes are used to set the size of the video displayed in the browser. The controls attribute instructs the browser to include basic video controls: pause, play and skip. The text “Video is not supported” will only be displayed if browser can’t load video.

To Add a link

<a

href=<https://www.wikipedia.org/>> This is a link to Wiki </a>

To add link that will pop up in another browser

<a

href=<https://www.wikipedia.org/>> This is a link to Wiki </a>

To add a link for contact information (the ./ tells the browser to look for the file in the current folder

<a href= “./index.html:> Contact <a/>

To add an image that acts like a link – this will open up a new browser

<a href="https://en.wikipedia.org/wiki/Opuntia" target="\_blank"><img src="#" alt="A red prickly pear fruit"/></a>

<ul>

   <li><a href="#media"> Media </a></li>

   <li><a href= "#habitat"> Habitat </a></li>

   <li><a href="#introduction">Introduction</a></li>

 </ul>

<table></table> creates a table

each <tr></tr> creates a row

<td></td> adds table data

This would show one row with two columns of data

<table>

 <tr>

   <td>73</td>

   <td>81</td>

 <tr>

</table>

<table>

 <tr>

   <th></th>

   <th scope="col">Saturday</th>

   <th scope="col">Sunday</th>

  </tr>

  <tr>

    <th scope="row">Temperature</th>

    <td>73</td>

    <td>81</td>

</tr>

</table>

The scope has two values

row – this value makes it clear that the heading is for a row

col – this value makes if clear the the heading is for a column

The colspan denotes the number of columns data can span acrss(out of town Monday-Wednesday)

<tr>

 <td colspan=”2”>Out of Town</>td>

<td>Back in Town</td>

<tr>

rowspan extends to multiple rows

<tr>

 <th>Morning</th>

   <td rowspan =”2”>Work</td>

   <td rowspan=”3>Relax</td>

</tr>

The<thead></head> is used to section off the tables heading

The <tbody></tbody> sections off the table body element

The <tfoot></tfoot> is used to create a for totals for example

<tfoot>

 <td>Total</td>

 <td>28</td>

</tfoot>

Forms are responsible for collecting information and sending it somewhere else

<form action=”/example.html” method=”POST”>

</form>

In the above example the action attribute determines where the information is sent and the method attribute is assigned a HTTP verb that is included in the HTTP request

To add text:

<form action=”/example.html” method=”POST”>

<input type=”text” name=”first-text-field”>

</form>

say you type in ‘important details’ in the text bar

When the form is submitted, the text: “first-text-field=”important details” is sent to /example.html

The label HTML tag is used to associate a text label with a form

<label for=”meal”> What do you want to eat?</label>

<input type=”text” name=”food” id=”meal”>

The number makes it so that you can only input numbers(side note: if you did type=”password” this would cover what your typing by using astericks or dots

<label for=”years”> Years of experience: </label>

<input id=”years” name=”years” type=”number” step=”1”</form>

This is how you input a range – and the smoothness on how you move around within that range

<label for=”volume”>Volume Control</label>

<input id=”volume” name=”volume” type=”range” min=”0” max=”100” step=”1”>

Checkbox input

<input id=”cheese” name=”topping” type=”checkbox” value=”cheese”>

#we are putting input first so that text is on the right

<label for=”cheese> Cheese</label>

radio – when we present multiple options but only want a user to pick one

<input type=”radio” id=”two” name=”answer” value=”2”>

<label for=”two”> 2 </label

Drop down list – if you have a whole list of options

Select creates a drop-down list

<label for=”lunch> What’s for lunch?</label>

 <select id=”lunch” name=”lunch>

   <option

value=”pizza”>Pizza</option>

 <option

value=”curry”>Curry</option>

<form>

<label for=”city”>Ideal city to visit?</label>

<input type=”text” name=”city” list=”cities” id=”city”>

<select

While <select> and <datalist> share some similarities, there are some major differences. In the associated <input>element, users can type in the input field to search for a particular option.

<form>

<label for=”city”>Ideal city to visit?</label>

<input type=”text” name=”city” list=”cities” id=”city”>

<datalist=”cities”>

 <option value=”New York City”> </option>

 <option value =”Tokyo”></option

<textarea> is used to create a bigger text box for blogs and posts

<label for=”blog”> New Blog Post:</label>

<textarea id=”blog” name=”blog” rows=”5 cols=”30”></textarea>

this creates a New Blog Post: title with 5 rows by 30 columns

To submit a form

<form>

<input type=”submit” value=”send”>

</form>

The required pattern states: that we the payment must be all numbers that are 14-16 characters long.

<input id=”payment” name=”payment” type=”text” required pattern=”[0-9]{14, 16}>

CSS

Inline Style: <p style="color: red;">I'm learning to code!</p>

The link element actually connects the CSS spreadsheet to the HTML document.

The link needs 3 different attributes:

Href – must be address to the CSS file. The path maybe a URL(

href=<https://www.codecademy.com/stylesheets/style.css>) or you if the file path is located on the same directory you can use href=”./style.css”

Rel – describes the relationship between the html file and the CSS file (ex: stylesheet)  
<link href="https://www.codecademy.com/stylesheets/style.css" type="text/css" rel="stylesheet">

<p class="brand">Sole Shoe Company</p>

To select this elements using CSS, we could use:

.brand {

}

CSS can select HTML elements by their id attribute. To select an id element, CSS prepends the id name with a hashtag (#)

an ID is meant to style only one element. As we’ll learn in the next exercise, IDs override the styles of tags and classes.

In HTML

<h1 class="title uppercase" id="article-title">

In CSS

.publish-time{

color: gray;

}

<h1 class="headline">Breaking News</h1>

h1 {

color: red;

}

.headline {

color: firebrick;

}

The color of the heading would be set to firebrick, as the class selector is more specific than the tag selector. If there was an ID attribute (and selector) were added to the code above the styles within the ID selector’s body would override all other styles for the heading

CHAINING SELECTORS: requiring an HTML element to have two or more CSS selectors at the same time.

H1.special {

}

This would only select the h1 elements that have a class of special

Nested elements: CSS also supports selecting elements that are nested within other HTML elements

<ul class='main-list'>

<li> ... </li>

<li> ... </li>

<li> ... </li>

</ul>

The nested <li> elements are selected with the following CSS:

.main-list li {

}

!important overrides even ID’s and should almost never be used

CSS declaration:

H1 {

Color: blue;

}

color: blue is the CSS declaration. Property = style (size, color) value = (18 px, size, color

TYPES OF CSS SELECTORS:

1. id selectors
2. class selectors
3. descendant selectors
4. chained or qualified selectors

font-family is the type of font. Should put the font face in quotes if its more than one word (font-family=”Times New Roman”)

font size

font-size: 18px;

font-weight: => this controls are bold or thin text appears

text-align: right, left, center;

TWO DISTINCTIONS OF COLOR:

Foreground color – the color an element appears in

Background color – styles elements background

Color: red

Background-color: black

Opacity: => 1.0 (100%) is fully visible while 0% is invisible

Background-image: url(“[www.example.com](http://www.example.com)”) or can use a path url(“images/mountains.jpg”)

The box model

P {

Height: 80px;

Width: 240px;

Borders: is a line that surrounds an element, like a frame around a painting. Boarders can be set with a specific width, style, and color

Width – the thickness. Can be set in pixels or thin, medium, or thick

Style – the design of the boarder

Color – color of border

P{

Border: 3px solid coral;

}

This border has a width of 3 pixels, a style of solid and color of coral

Border-radius: 5px; => equivalent to a circle with radius 5 pixels border-radius: 100%; => this is a circle

Padding: 10px; => puts 10 pixels of space between the content of the paragraph (the text) and the borders, on all four sides

1. padding-top
2. padding-right
3. padding-bottom
4. padding-left: 10 px; => contents of left side will have padding of 10 pixels

padding: 6px 11px 4px 9px; => top, right, bottom, left

padding: 6px 10px; => the 6 is for top and bottom and 10 is for left and right

margin refers to space directly outside of the box

1. margin-top
2. margin-right
3. margin-bottom
4. margin-left

margin: 6px 10px 5px 12px

margin: 6px 12px

div {

margin: 0 auto;

} => this centers in their containing element

but in order to fully center an element, a width must be set for that element.

Unlike horizontal margins, vertical margins do not add. Instead, the larger of the two vertical margins sets the distance between adjacent elements.

p {

min-width: 300px;

max-width: 600px;

}

p {

min-height: 150px;

max-height: 300px;

}

hidden - when set to this value, any content that overflows will be hidden from view.

scroll - when set to this value, a scrollbar will be added to the element’s box so that the rest of the content can be viewed by scrolling.

visible - when set to this value, the overflow content will be displayed outside of the containing element. Note, this is the default value.

P {

Overflow: scroll;

}

how to have a CLEAN SLATE

\* {

margin: 0;

padding: 0;

}

visibility: hidden or visible;

\*{

box-sizing: border-box;

}

This code resets the box model to border-box for all HTML elements

Position: relative;

Top: 20px;

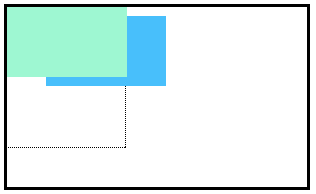
Left: 50px;

* This value allows you to position an element *relative* to its default static position on the web page. And this div will be moved down 20 pixels and to the right 50 pixels from the default static position. These are OFFSET properties

ABSOLUTE: When an element’s position is set to absolute all other elements on the page will *ignore* the element and act like it is not present on the page. The element will be positioned relative to its closest positioned parent element.

Position: relative;

Position: fixed; => this will fix the div while you scroll



.box-top{

z-index: 2;

}

.box-bottom{

z-index: 1;

}

The default display for some tags, such as <em>, <strong>, and <a>, is called *inline*. Inline elements have a box that wraps tightly around their content, only taking up the amount of space necessary to display their content and not requiring a new line after each element.

H1 {Display: inline;} => this is to turn all H1 elements into inline

BLOCK DISPLAY – these elements fill the entire width of the page by default, but their width property can also be set. Examples: <h1> through <H6>, <p>, <div>, <footer>

INLINE-BLOCK is a combination of both

.rectangle {

display: inline-block;

width: 200px;

height: 300px;

} => they will be on the same line if they can fit

float: right; => this will move an element as far right as possible

CLEAR:

1. Clear: left; — the left side of the element will not touch any other element within the same containing element.
2. right — the right side of the element will not touch any other element within the same containing element.
3. both — neither side of the element will touch any other element within the same containing element.
4. none — the element can touch either side.

COLOR:

H1{

Color: rgb(23, 45, 23);

}

 0 to 255. The first number represents the amount of red, the second is green, and the third is blue

color: (hsl(120, 60%, 70%) => first number represents the degree of the hue, can go from 0 to 360. The 2nd and 3rd numbers are representing the saturation and lightness respectively.

OPACITY: color: hsla(34, 100%, 50%, 0.1); => The fourth value is the alpha. This last value is sometimes called the opacity. If alpha is 0 it will be fully transparent

Color: rgba(234, 45, 97, 0, 0.33); alpha can only be used with the HSL and RGB colors

Font-family:

Font-weight:

Font-style:

Word-spacing: 0.3em; => word spacing

Letter-spacing:

Text-transform: uppercase;

Line-height: => value can be unit-less number like 1.2 or a unit (12px, percents, ems, or rems)

h1 {

font-family: "Garamond", "Times", serif;

} => if Garamond is not available use times, if both aren’t use any serif

to bring a font from the web browser

<head>

<link href="https://fonts.googleapis.com/css?family=Droid+Serif" type="text/css" rel="stylesheet">

</head>

OR you can take link <https://fonts.googleapis.com/css?family=Space+Mono:400,700> into browser You will need to focus on the rules that are directly labeled as /\* latin \*/.

GRIDS

COLUMNS

We can define the columns of out grid by using the CSS property grid-template-columns: 100px 200px; => NUMBER OF COLUMNS: 2, the first column is 100px wide and 2nd is 200px wide.

Can also do percentage

Width: 1000px

Grid-template-columns: 20% 50% => first column is 20% of 1000

ROWS

Height: 500px;

Grid-template-rows: 10% 20% 600px;

The first row is 50 pixels tall (10% of 500

BOTH

Grid-template: 200px 300px / 20% 10% 70%

Values before / determines the rows and the values after / is the columns

grid-template: 1fr 1fr 1fr / 3fr 50% 1fr; => taking a fraction rows all of equal space

grid-template-columns: repeat(3, 100px);

Using the repeat function will make the grid have three columns that are each 100 pixels wide

Grid-template-columns: repeat(2, 20px 50px)

first and third columns will be 20 pixels wide and the second and fourth will be 50 pixels wide.

grid-template-columns: 100px minmax(100px, 500px) 100px;

}

1nd and 3rd column will always be 100px, the 2nd column will be in between 100-500px

grid-gap: 20px 10px; will set the distance between rows to 20 pixels and the distance between columns to 10 pixels

.item {

grid-row-start: 1;

grid-row-end: 3;

}

the element of class item will take up two rows in the grid, rows 1 and 2

grid-column: 4 / span 2;

.item {

grid-area: 2 / 3 / 4 / span 5;

}

1. grid-row-start
2. grid-column-start
3. grid-row-end
4. grid-column-end

grid-template-columns: repeat(3, 400px);

justify-items: center;

}

 the .card <div>s will be centered inside of their columns. They will only be as wide as necessary to contain their content

We can use justify-content to position the entire grid along the row axis.

align-items is a property that positions grid items along the block, or column axis.

Grid-auto-rows: => this implicitly returns more rows if you need more.

grid-auto-flow specifies whether new elements should be added to rows or columns.

* Grid-auto-flow: row — specifies the new elements should fill rows from left to right and create new rows when there are too many elements (default)

JAVASCRIPT

Adding quotes = the \ tells Javascript to ignore the character’s special meaning and just use the literal value of the character. Ex. “the man whispered \”please speak to me.\””

Javascript implicit coerces integer into string or vice versa “hello” + 1 => “hello1” “1” == true =>  true because it changes 1 to true

Strict equality- ===    “1” === 1 is false but “1” == 1 is true

var price = 15.00

var money = 20.00

if(money>= price) {

 console.log("buy the hammer")}

else {

 console.log("dont buy the hammer")

}

//the curly braces is the code that is executed given the condition

var weather = "sunny";

if (weather ==="snow") {

 console.log("bring a coat")

} else if (weather ==="rain"){

 console.log("bring a rain jacket")

} else {

 console.log("wear what you have on")

}

Falsy values: “”, null, undefined, 0, NaN

Truthy values Examples: true, 42, “pizza”, “0”, “null”, “undefined”, {},[]

To use the ternary operator, first provide a conditional statement on the left-side of the ?. Then, between the ? and : write the code that would run if the condition is true and on the right-hand side of the : write the code that would run if the condition is false

var isGoing = true;

var color = isGoing ? "green": "red";

console.log(color)

A switch statement is another way to chain multiple else if statements that are based on the same value without using conditional statements. Instead, you just switch which piece of code is executed based on a value.

The else if statement (option ===[value]) has been replaced with a case clause (case:[value]) and those clauses have been wrapped inside the switch statement

Var option = 3

Switch (option) {

   Case 1:

Console.log(“you selected option 1.”);

Break;

   Case 2:

Var tier = “none”;

Var output = “youll receive”;

Switch(tier){

 …

  default:

output += “one copy”

}

default is what’s automatically added if nothing else matches the case

WHILE LOOPS

var start = 0; // when to start

while (start < 10) { // when to stop

 console.log(start);

 start = start + 2; // how to get to the next item

}

FOR LOOPS

for (var i = 0; i < 6; i += 1){

  console.log(“Printing out i = “ + i);

You can even set a function as a variable

Var catSays = function(max){

put function here

}

you can just put function since you already named it, otherwise itd be redundant

HOW TO DEFINE AN OBJECT

Create a variable and assign it to {}

Something an object can do is a method

// function expression catSays

var catSays = function(max) {

 var catMessage = "";

 for (var i = 0; i < max; i++) {

   catMessage += "meow ";

 }

 return catMessage;

};

// function declaration helloCat accepting a callback

function helloCat() {

 return "Hello " + catSays(3);

}

// pass in catSays as a callback function

console.log(helloCat(catSays));

SPLICE

Arr = [1,2, 3, 4]

Arr.splice(1, 2, “Hello”, “there”)

This starts at index[1] and deletes two elements, and add “hello” and “there”

* [1, “Hello”, “There”, 4]

ENNUMERABLE forEach

var donuts = ["jelly donut", "chocolate donut", "glazed donut"];

function printDonuts(donut){

 donut += " hole";

 donut = donut.toUpperCase();

 console.log(donut);

}

donuts.forEach(printDonuts)

but could express as an inline function like:

donuts.forEach(function(donut){

 donut += " hole"

 donut = donut.toUpperCase();

 console.log(donut)

})

forEach(element, index, array)

MAP alters the original array and returns a new one

var donuts = ["jelly donut", "chocolate donut", "glazed donut"];

var newer = donuts.map(function(donut, i, arr) {

 donut += " hole";

 donut = donut.toUpperCase()

 return donut

});

console.log(newer)

addTodo() shows what’s inside the circle and addTodo shows the actual function

this is referring to the entire object (everything inside the curly braces}

{

name: 'Gordon',

sayName: function(){

console.log(this);

}

}

forEach(callback, this) => this will refer to the review objects

This is all inside of an object

addTodo: function (todoText){

this.todos.push({

todoText: todoText,

//the first todoText is always going to be todoText, while the second todoText is the //is the parameter which is whatever you pass in. so if you do addToDo(“hi”), then // it will be todoText: “hi”

completed: false

});

var gordon = {

name: 'Gordon',

sayName: function(){

console.log(this.name)

}

}

gordon.sayName() => Gordon

putting a function on an object is a method

so say name is a method on the Gordon object

how to add (x), when an object is completed

console.log("My Todos");

for (var i = 0; i < this.todos.length; i++){

// check if .completed is true

if(this.todos[i].completed === true){

//print with(x)

console.log('(x)', this.todos[i].todoText)

} else {

console.log("( )", this.todos[i].todoText)

}

}

{} === {} => false because its comparing the addresses of the objects. It will only hold true if your talking about same object (same house)

//is a method, everytime someone clicks, it will run the function

displayTodosButton.addEventListener('click', function(){

})

Acts like a timer

setTimeout(function() {}, 5000)

your using 5000 because it runs in milliseconds

runWithDebugger(function(){

for (var I = 0; I < 10; i++){

console.log(i);

}

});

The highlighted portion is the callback function and nonhighlighted is the higher order function

BIND

Returns a copy of the function where ‘this’ is set to the first argument passed in .bind()

Apply and Call will change the this argument inside of a function and apply it immediately. Whereas bind you have to run an additional step

Var explicitlySetLogThis = logThis.bind({name: ‘Gordon’})

// then had to run again

explicitlySetLogThis(); => {name: ‘Gordon’}

logThisWithArguments.apply({name: 'gordon'}, ['hi', 'gordon']);

logThisWithArguments.call({name: 'gordon'}, 'hi', 'gordon' ); //dont need to put into //array

Math.random() => creates a decimal between 0 and 1 but not quite either

Math.floor() => rounds the number down

parseInt() => converts a string to an integer

var a = parseInt(“11”, 2) => this converts 11 to base 2

When you declare a variable with the varkeyword, it is declared globally, or locally if declared inside a function.

The let keyword behaves similarly, but with some extra features. When you declare a variable with the let keyword inside a block, statement, or expression, its scope is limited to that block, statement, or expression.

Const has all the awesome features that let has, with the added bonus that variables declared using const are read-only. They are a constant value, which means that once a variable is assigned with const, it cannot be reassigned.

Object.freeze(obj) => this freezes an object so you can’t change it

const myFunc = function() {  
  const myVar = "value";  
  return myVar;  
}

is equivalent to

const myFunc = () => {  
  const myVar = "value";  
  return myVar;  
}

is equivalent to:

const myFunc = () => “value”

this is a function that doubles all the items.

const = doubler = (item) => item \* 2

FBPosts.filter((post) => post.thumbnail !== null && post.shares > 100 && post.likes > 500)

var words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];

const result = words.filter(word => word.length > 6);

console.log(result);

// expected output: Array ["exuberant", "destruction", "present"]

const squareList = (arr) => { "use strict"; const squaredIntegers = arr.filter( (num) => num > 0 && num % parseInt(num) === 0 ).map( (num) => Math.pow(num, 2) ); return squaredIntegers; };

For default values

function greeting(name = "Anonymous") {  
  return "Hello " + name;  
}  
console.log(greeting("John")); // Hello John  
console.log(greeting()); // Hello Anonymous

To pass an a variable amount of arguments

const arr = [6, 89, 3, 45];  
const maximus = Math.max(...arr); // returns 89

REST OPERATOR

Function multiply(multiplier, …theArgs) => this can pass an any amount elements in this function

var voxel = {x: 3.6, y: 7.4, z: 6.54 };  
var x = voxel.x; // x = 3.6  
var y = voxel.y; // y = 7.4  
var z = voxel.z; // z = 6.54

Here's the same assignment statement with ES6 destructuring syntax:

const { x, y, z } = voxel; // x = 3.6, y = 7.4, z = 6.54