**Youth - Detailed Day Outline**

# Day 1 (Jan 13/14):

* Start off with introductions:
  + What is your name?
  + Fun fact?
  + Any hobbies (video games, playing sports, etc.)?
* Tell them what I’ll be teaching:
  + Basics of HTML
  + Basics of CSS
  + Basics of JavaScript
  + And then we make a small game
* Tools to use
  + Repl.it - a website based code editor that allows you to write code without having to download anything
    - If they DON’T have repl.it or can’t make an account, then using a basic text document (.txt) works (until we get to JS)
    - Search “Notepad” for Windows, “TextEdit” for Apple
* Basics of HTML
  + Tags < >
    - When writing HTML, we use angle brackets to define the type of element we want to create
  + <!DOCTYPE html>
    - a.k.a. Document Type Definition
    - Always at the top of the page (like the very top)
    - Defines the type of HTML being used
    - There are 5 different types of HTML versions. The one we are using is HTML5, the most recent version
  + <html> </html>
    - Defines that we are using HTML and not any other web-based language
    - Everything we write must go in between these tags
  + <head> </head>
    - This is the head, or brain, of the document. Just like a human body, the head stores all the information such as your name, the things you like or dislike, and other objects like your age, the school you attend, and where you live.
    - <title> </title>
      * Found in the head, it is what that website page is called. Just like how you have a name, so does this specific page.
    - <meta charset=”utf-8”>
      * Also found in the head section, this charset lets us use almost any keyboard key that we want. Some computers may not take specific keys so it’s important to add an object that allows us to use almost any keyboard key to avoid any confusion
  + <body> </body>
    - Just like your body which has your torso and arms (hopefully), the website’s body also has content.
    - Headings
      * Goes from heading 1 to 6 and defined by <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, these headings are used in various ways to make content stand out in a website. For example, using a <h1> will make whatever text that is stand out the most as it is the largest
    - <p> </p>
      * Paragraph elements contain text that describes what’s being shown on the website. Other elements such as images can be contained within the paragraphs
    - <i> </i>
      * Italic element
      * Can be placed in any text element such as the headings or paragraphs
    - <b> </b>
      * Bold element
      * Can be placed in any text element such as the headings or paragraphs
    - Lists
      * Unordered list defined by <ul> </ul>
      * Ordered list defined by <ol> </ol>
      * Within these different types of lists, we use <li> </li> to insert a bullet point
    - Anchor elements
      * Shown by using <a> and ending with </a>
      * We use anchor elements as hyperlinks
        + Example: <a href=”google.ca”>
        + When I click on the link on my HTML page, it will take me to the website that I have linked it to
      * You can also use anchor elements to link to different parts of your website
    - Images
      * To enhance your website, we may want to add images. Visual aids are really helpful to someone who may be viewing your website.
      * <img src=”cat.jpg” alt=”cat” width=”100” height=”100”>
        + Src: where the image is located in our files
        + Alt: the text associated with the image. Sometimes the image might not load or a person with visual disabilities may have problems with looking at the image so adding extra (BUT SHORT) text that goes with the image helps
        + Width: how wide the image is
        + Height: how tall the image is
    - Tables
      * Start off with creating a <table> </table>
      * Within each table, there are table rows <tr> </tr>
      * Within table rows are <td> </td>
        + “Table data”
      * Why would we want to use tables?
        + Maybe to organize any data we have such as:

A schedule

Weather

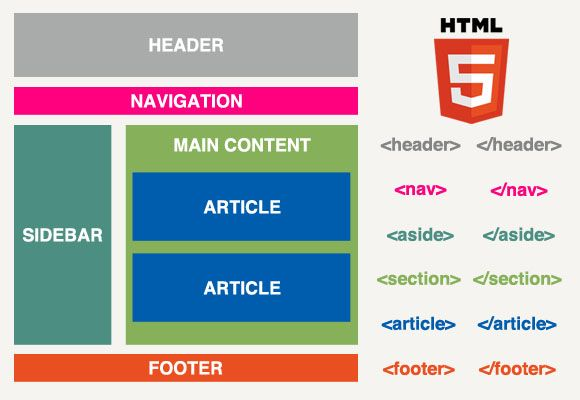
Finances

# Day 2 (Jan 20/21):

**Structuring our HTML document**

Currently, we’ve only used text and image elements such as paragraphs, headings, links, and images. HOWEVER, that's not helpful when we’re trying to organize things so let's make it nicer.

* <header> </header>
  + Any guesses as to what's the difference between header and head???
    - Head: stores information
    - Header: helps with navigation, contains company logo, search form, etc.
* <nav> </nav>
  + The navigation bar
  + Used to get to different parts of your website
* <main> </main>
  + The “body” or “beefy” part of your HTML file
  + Known as the “main” because it usually contains the majority of the information on the webpage (a.k.a. Dominant body)
* <footer> </footer>
  + At the end of your webpage
  + Usually contains a website logo, social media links, terms and conditions, etc.

[](https://codelikethis.com/lessons/www/html5)

These are called SEMANTIC tags. A SEMANTIC tag is any tag or element that has a specific name or property associated with it. For example, the “p” element refers to a paragraph element and a “img” element is associated with an image. On the other hand, an element like a “div” is not a semantic element because it has no specific meaning.

**Cascading Style Sheets**

* Also known as CSS, they help make our websites look nice and pretty by changing colours, adding fonts, etc.
* Four types of ways to input CSS
  + Inline styles
    - Directly edit HTML tags that change the style of an object
    - E.g. <p color=”red”>This paragraph is now red</p>
  + Embedded styles
    - Found in the “head” section
    - Written in between <style> </style>
  + External styles (the ones we will be focusing on)
    - Create a new file that ends with .css
    - E.g. word.css, styles.css, etc.
  + Imported styles (we will not focus on this)

**How to use CSS**

* Within an embedded or external style, we want two things. Any guesses????????
  + The tag (p, body, main, etc.) AND brackets ({ })
  + E.g. body { }

Unlike HTML which has a rigid outline of what we may want to make, CSS is more flexible in terms of creativity for your website. Because of this, CSS can be much trickier especially the more in-depth you get in web development. Remember:

**HTML is for content, CSS is for styling**

* Color: changes the color of the text
* Background-color: changes the background color of the tag/object
* Font-size: changes font size of a tag
* Font-style: can change text style (italics, bold, etc.)
* Font-family
* Text decoration

Let’s create a styling:

Body { color: yellow; }

* Body is the **selector**
* Color is the **property**
* Yellow is the **value**
* Remember to add semicolons!!!

**Classes and IDs**

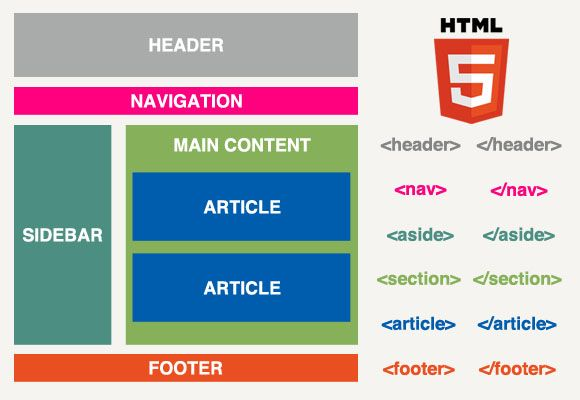
* Imagine a school. You have the following:
  + Class(es)
  + id(s)
  + What's the difference????????
    - Answer: Classes are for EVERYONE and ids are for a single person
  + Class
    - Denoted by .class
    - Can be used as many times as we want as long as we specify what element uses that tag
    - E.g. <h1 class=”top”> </h1> <p class=”top”> </p>
    - .top { font-size: 200%; color: green; }
  + Ids
    - Denoted by #id
    - Unlike classes, they can only be used for one element
    - E.g. <h2 id=”bottom”> </h2>
    - #bottom { background-color: blue; font-style: italic; }

**Flex and Grid**

Flex, also known as flexbox, is used for **one-dimensional layouts** like columns (e.g. mobile device like a phone) whereas a grid is better used for larger screens with **two-dimensional layouts** like columns and rows.

**Flex**

* Use “display: flex;”
* Can use flex for horizontal or vertical layouts
* HORIZONTAL
  + Flex-direction: row;
* VERTICAL
  + Flex-direction: column;

[](https://codelikethis.com/lessons/www/html5)

**Grid**

* Usually more complicated than a flexbox as it has a lot more involved in it
* Use “display: grid;”
* A grid has COLUMNS and ROWS
  + In between these columns and rows, you can insert a gap
  + “Gap: 50px;” or “grid-gap: 50px”
* How to create a responsive grid in CSS
  + In body, input the following:
    - Display: grid;
    - Grid-columns: 1fr 3fr;
    - Grid-template-rows: auto auto auto
      * Why do you think we have 3 autos?????
        + Answer: Because we will have 3 rows
    - And finally, grid-template-areas:
      * “Header header”
      * “Nav nav”
      * “Aside main”
      * “Footer footer”
      * What do you think that will do????
        + Answer: Set our tags to be in the HTML arrangement we want using CSS (just like our picture)
    - However, we need to tell our individual CSS elements that we’re using a grid for our layout
      * Header { grid-area: header; }
      * Main { grid-area: main; }
      * Aside { grid-area: aside; }
      * Nav { grid-area: nav; }
      * Footer { grid-area: nav; }
    - Finally, we can’t really see the grid so lets add some borders to it!!!
      * Header, main, aside, nav, footer { border: 1px solid black; }

# Day 3 (Jan 27/28):

## **Review**

HTML (HyperText Markup Language) is the content of our website

* All HTML tags MUST be in “angle brackets” < >
* The majority of the code in HTML must have opening and closing tags. Closing tags have a slash in between the first angle bracket and the element (e.g. Opening: <p>, Closing: </p>
* Every HTML document must have <!DOCTYPE html> at the top to show that we are using HTML
* After that, all code must be in-between the <html> </html> tags

CSS (Cascading Style Sheets) is the styling (or design) of our website

* A CSS element consists of the following:
  + Selectors: the HTML tag we are applying the style to (e.g. p, body, header, etc.)
  + Property: the part of the HTML tag we are going to change (e.g. color, font-family, font-weight, etc.)
  + Value: the specific property that we are applying (e.g. green, “Arial”, bold, etc.)
* We also have classes and ids
  + Classes start with a dot (.) whereas ids start with a hashtag (#)
  + Classes can be applied to as many HTML elements as we want
  + Ids can only be applied to one HTML element
* Comments in HTML and CSS can be written as followed: /\*\* COMMENT \*\*/
  + Comments do not affect the code of these documents

## **JavaScript**

* The “Development” aspect of Web Design and Development
* Javascript is the behaviour of our website
* To connect a Javascript file to our HTML file, we write “<script src="myscripts.js"></script>”
* Basics of JavaScript
  + Data types
    - We have a variety of different types of data in Javascript but the ones we will be focusing on are:
      * Boolean
      * Number
      * String
      * Undefined
    - “console.log();”
      * Using this code in Javascript will print something to our “console”
      * Interface to execute simple lines of code
      * Ends in a semicolon
        + Semicolons are optional in Javascript
    - Boolean
      * Used to check if something is “true” or “false”
      * Basically a yes/no question
      * console.log(5 > 1);
        + If this statement is correct, it will print “true”
      * console.log(3 > 5);
        + Will print false
    - Number
      * Used to store a numerical value
      * Can use decimals and integers
      * console.log(5 + 7.2);
    - String
      * Used for any text value
      * In HTML, we had text values within our headings and paragraphs. The same can be applied when using Javascript
      * All text values must be within quotation marks
      * console.log("I am learning Javascript"); vs console.log(I am learning Javascript);
    - Undefined
      * We assign “undefined” to any variable or Javascript code that DOES NOT have anything assigned to it
      * Maybe we don’t know if we want to assign a part of the code as a Number or as a String; instead, we will say it “undefined”
      * Example: I might want to store a date as a Number or a String, but I don’t know which one so I’ll store it as undefined
  + Variables
    - Use “var” or “let” to define a variable (I’ll be using “let”)
    - Making a boolean
      * let isDaytime = true;
        + Let: variable
        + isDaytime: variable name
        + True: data type (in this case, boolean)
    - Making a number
      * let minutes = 60;
      * let seconds = 60;
    - Making a string
      * let bookTitle = “The Hobbit”
      * let addition = “6.7 + 32”
        + Why does this not add up?

Doing “6.7 + 32” will make it a string so BE CAREFUL

* + Using operators
    - We have an arrangement of operators in Javascript to do calculations
    - Arithmatic: +, -, /, \*, %
      * Basic computations: Plus, minus, times, divide
      * Modulus operator (%): gets the remainder of our operation
        + E.g. 10%4 = 2
        + 11%11 = 0
        + 15%2 = 1
    - Comparison: >=, ==, <=, <, >, ===
      * Greater or less than signs: >, <
      * Greater than or equal to: >=, <=
      * Equals: ==
      * Equals comparison plus data type comparison: ===
        + What is the difference between “1”==1 vs. “1”===1?

# Day 4 (Feb 3/4):

## **Functions**

* A block of code that is used for a specific task
* Need the following:
  + “Function” keyword
  + Function name: functionName
  + Parameter brackets: () (parenthesis)
  + Curly brackets for code { }
* A function must be called for it to run. This means that there must be a specific button or feature in your website that gets the function to compile its code
* Parameters in functions
  + These are the requirements we need for a function to run and parameters are written in the form of a variable name
  + E.g. function functionName(parameter);
  + Parameters can be used in numerous ways:
    - Solving a math problem using numbers
    - Getting someone’s name using strings
    - Answering questions using booleans
* Return
  + The “return” keyword can be used to end a function
  + Typically put at the end
  + Can return an operation, variable or a data type
    - E.g. return “word”, return true, return isDaytime;
  + Assigning a variable to a function can assign whatever return value that is in the function to the variable
    - Let currentTime = getTime(120);

## **Calling a Function**

* Type the function name with the parameters (if it has no parameters, just have the parenthesis)
  + E.g. functionName() or functionName(parameter)
* We can call a function to the console by assigning a variable to it. If the function has any console.log code, then that code will be printed out

## **Checking Multiple Conditions**

* When we want to check multiple conditions, we use “&&” or “||”
  + &&: both conditions must be satisfied for the operation to work
  + ||: one of the conditions must be satisfied for the operation to work
* Booleans
  + console.log(true && false); //returns false
  + console.log(true && true); //returns true
  + console.log(true || false); //return true
* Strings
  + console.log(“cat” && “dog”); //returns “dog”
  + console.log(“cat” || “dog”); //returns “cat”
* Numbers
  + console.log(5 > 0 && 2 < 0); //returns false
  + console.log(5 > 0 || 2 < 0); //returns true

## **If-Statements**

* We use if-statements to check if a condition is true or not
* We denote this via if(parameter) followed by curly brackets like so:
  + if(parameter { code }
* If our first statement isn’t true, we can have another condition by adding an “else if” which is followed after the curly brackets
  + else if(parameter)
* If nothing gets through our if and else if statements, then we can call an “else” condition
  + Else { }
  + An else statement is our default condition if nothing gets through our if statements
* Example:
  + if(hours == 1) { console.log(“Good morning”); } else if(hours == 2) { console.log(“Good afternoon”); } else if(hours == 3) { console.log(“Good night”); } else { console.log(“Good”); }

## **Loops**

A loop is a coding mechanism in which you can repeat a task multiple times without having to rewrite it

### **For-Loop**

* Start off with a “for” and have the parameters next to the loop like so: for(let i = 0; i < 5; i++)
* In the parameters, you need the following:
  + The starting condition: e.g. let i = 0;
  + The ending condition: e.g. i < 5;
  + The iteration count (how many times your starting condition will change for every loop): e.g. i++ which means “for every loop, the variable ‘i’ will increase by 1”
* Curly brackets are followed after every loop
  + Within the curly brackets is the code you want to write

### **While Loop**

* Start with a “while” and have your parameter
  + Your parameter is a specific condition such that the loop will continue while your parameter is true
  + E.g. while(minutes < 60) { minutes += 1; }

### **Do-While Loop**

* Basically the same thing as a while loop but its “do { } while(parameter)”

### **Differences**

* for - loops through a block of code a number of times
* while - loops through a block of code while a specified condition is true
* do/while - also loops through a block of code while a specified condition is true
* Also, a while loop will check the parameters first and then execute the code if it is true while a do/while loop will run the code first and then check if the parameters are true

### 

## **Assignment #1**

Create a function called “repeatNumbers” that takes a number as its parameter and prints out “[NUMBER] is divisible by 2” if a number can be divided by 2, prints out “[NUMBER] is divisible by 5” if a number can be divided by 5, AND if a number is divisible by 2 and 5, it must print out “[NUMBER] is divisible by 2 and 5” where [NUMBER] is a number at a specific point.

Time limit: 20 minutes

# Day 5 (Feb 10/11):

## **Math and Random in JS**

Being able to do calculations in a program is a big thing, and typing out an entire equation can be tedious. Instead of doing that, we can use the “Math” library:

* Math.PI gets us the numbers for PI
* Math.round(x) rounds to the nearest integer
* Math.ceil(x) always rounds up to the nearest integer
* Math.floor(x) always rounds down to the nearest integer
* Math.pow(x, y) gets two parameters: x being the number being multiplied and y being the power
* Math.sqrt(x) gets the square root of that number
* Math.random() will give us a number between 0 and 1 (0 <= Math.random() < 1)

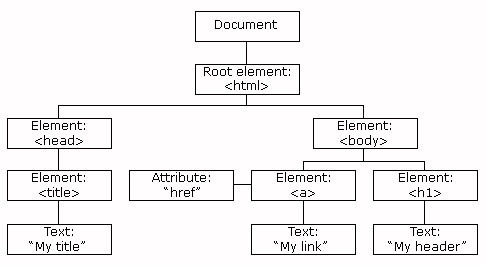
### 

# Day 6 (Feb 17/18):

## **Document Object Model**

The Document Object Model (or DOM) is a way to change HTML and CSS elements on a website by using Javascript. We can do the following:

* Remove or add HTML elements (or text)
* Remove or add CSS stylings to an element
* Change the way a website functions



Note: Just a reference of what it can look like

In the index.html file, make sure you have the keyword “async” or “defer” right after your source file.

Keywords used to get the DOM in JS:

* document: gets the webpage we’re working with
* getElementById(“element”): looks for the element with the same ID
* innerHTML: grabs the HTML of that element and can change it accordingly

Example:

* document.getElementById(“demo”).innerHTML = “We are using the Document Object Model to change this element through Javascript”;
* This will change the element with the ID of “demo” to the text of “We are using the Document Object Model to change this element through Javascript”
* Add a function called “callDemoFunction()” and assign it to a button

### **Event Listeners**

Using event listeners, we can check if a user is making an action on our website. When that happens, we can write some code that lets us know that a change is happening. First, we need to do a few things:

* Create a HTML button, assign an ID called “event” to it, and then create a variable that is connected to that HTML element through its ID
* Then, we assign that variable to addEventListener(x, y);
  + addEventListener has two parameters: x which is the type of response we are looking for (click, Enter key, spacebar, etc.) and y which is the function that runs the code

Code for that:

* Let testingEventListener = document.getElementById(“event”);
* testingEventListener.addEventListener(“click”, function() { alert(“Event has been clicked”) }
* <button id=”event”>Click me!</button>

### **Using Event Listener to Add Numbers**

What if we want user inputs? We have numerous ways to do that. Let’s look at one approach:

* First, create a button called “Add Numbers.” When this button is pressed, it will give us a prompt()
  + A prompt is a popup that asks the users a question for them to answer
  + We can attach a variable to a prompt so that the user’s input is set to that variable like so:
    - let question = prompt(“What is your favourite colour?”);
    - A prompt will always result in a **string** data type so we need to convert that into an integer
* Second, we need to make a paragraph that will display our sum and attach an id to that
* Third, we’ll make an event listener that attaches to our button. When we click it, it will prompt us for two numbers to add and then display the sum in our paragraph element.

## **Assignment #2**

Create a function called “addNumbers” that takes two numbers as its parameter and returns the sum of those two numbers. Also, we need to use something called “parseInt()” which changes a variable to a number. You can use it like this:

* let numberAsString = “10”
* let numberAsInteger = parseInt(numberAsString);

Time limit: 10 minutes

### **Quick Review of JavaScript**

* The “Development” aspect of Web Design and Development
* Javascript is the behaviour of our website
* To connect a Javascript file to our HTML file, we write “<script src="myscripts.js"></script>”
* Basics of JavaScript
  + Data types
    - We have a variety of different types of data in Javascript but the ones we will be focusing on are:
      * Boolean
      * Number
      * String
      * Undefined
  + Variables
    - Use “var” or “let” to define a variable (I’ll be using “let”)

#### **Functions**

* A block of code that is used for a specific task
* Need the following:
  + “Function” keyword
  + Function name: functionName
  + Parameter brackets: () (parenthesis)
  + Curly brackets for code { }
* A function must be called for it to run. This means that there must be a specific button or feature in your website that gets the function to compile its code
* Parameters in functions
  + These are the requirements we need for a function to run and parameters are written in the form of a variable name
  + E.g. function functionName(parameter);
  + Parameters can be used in numerous ways:
    - Solving a math problem using numbers
    - Getting someone’s name using strings
    - Answering questions using booleans
* Return
  + The “return” keyword can be used to end a function
  + Typically put at the end

#### **Calling a Function**

* Type the function name with the parameters (if it has no parameters, just have the parenthesis)
  + E.g. functionName() or functionName(parameter)
* We can call a function to the console by assigning a variable to it. If the function has any console.log code, then that code will be printed out

#### **If-Statements**

* We use if-statements to check if a condition is true or not
* We denote this via if(parameter) followed by curly brackets like so:
  + if(parameter { code }
* If our first statement isn’t true, we can have another condition by adding an “else if” which is followed after the curly brackets
  + else if(parameter)
* If nothing gets through our if and else if statements, then we can call an “else” condition
  + Else { }
  + An else statement is our default condition if nothing gets through our if statements
* Example:
  + if(hours == 1) { console.log(“Good morning”); } else if(hours == 2) { console.log(“Good afternoon”); } else if(hours == 3) { console.log(“Good night”); } else { console.log(“Good”); }

#### **For-Loop**

* Start off with a “for” and have the parameters next to the loop like so: for(let i = 0; i < 5; i++)
* In the parameters, you need the following:
  + The starting condition: e.g. let i = 0;
  + The ending condition: e.g. i < 5;
  + The iteration count (how many times your starting condition will change for every loop): e.g. i++ which means “for every loop, the variable ‘i’ will increase by 1”
* Curly brackets are followed after every loop
  + Within the curly brackets is the code you want to write

## **BUILDING A GAME (Rock Paper Scissors)**

### **Starting the HTML Layout**

First, we need to build the HTML layout of the game. We’re going to be using the following items:

* Header and footers
* Main element
* Buttons
* Headings
* And other text elements

Let’s build a new Repl.it workplace and call it “Rock paper scissors.” Once you’ve done that, move the script tag you see right before the end of the body and move it to the <head> section. Also, delete the “Hello world” text in the body. Once you’ve done that, build the following:

* Header
  + H1 with the words “Rock paper scissors!”
  + And a horizontal line (<hr>)
* Main
  + Within the main, we’re going to need a few things
    - An <h2> element called “Score”
    - Two <h3> elements called “Player” and “Computer”
    - Two <h4> elements set to 0. One <h4> element will have an id called “playerScore” and the other have an id of “computerScore”
    - Also, we want some text that says what moves each player made
    - Three buttons. Each button will have the move you can make in the game (Rock, Paper, and/or Scissors) and will have their own ids.
  + In between the h3 and paragraph elements, create a “div” that is connected to a class called “grid”
  + Also, in between the buttons, create a “div” that is connected to a class called “flex”
* Footer
  + In the footer, we will have another horizontal line and a paragraph element that says “&copy; [your name]’s Game!”

# 

# Day 7 (Feb 24/25):

## **BUILDING A GAME (Rock Paper Scissors)**

### **Adding Styles with CSS**

Before we add our stylings, we need to set the font family to all HTML elements. To do that, we call “html” in CSS and add a font family of ‘Lucida Sans’ to it.

First, we need to add some styles. We are going to add the following:

* Header and footer
  + Background colour of “palevioletred”
  + Padding of 1em
    - This gets all elements within the header and adds some space between the borders of the header and the elements within
  + In addition, our text is aligned to the left. We don’t want that. Instead, let’s move it to the center. To do that, call “text-align: center;”
* Main
  + Background colour of “aliceblue”
  + Padding of 1em
  + Get the h2 of main and align the text to the center
* Main .grid
  + Here, we’ll create a grid by doing “display: grid” and create two columns: “grid-template-columns: auto auto;”
  + We also want to move the text to the center
* Main .flex
  + Display: flex;
  + Flex-direction: row;
  + Add a gap of 2em to separate the buttons
  + Add padding of 1.25em
  + Finally, use “justify-content: center” to align the items towards the center
* Buttons
  + font-weight: bold;
  + Padding: 1em;
* Horizontal line
  + Notice how our horizontal lines are now white. Instead, we want them to be black. We’re going to write the following CSS stylings:
    - Border-style: solid;
    - Border-width: 1px;
    - Border-color: black;
* Body
  + Currently, we see white lines on the left and right sides of the website. That’s not something we want and doesn’t look nice. To fix that, we need to call the “body” and assign a margin of 0 to it.

### **Making a Functional Game with JavaScript**

The first thing we need to do is get all of the IDs attached to our HTML elements. Here are the following IDs:

* let rock = document.getElementById("rock");
* let paper = document.getElementById("paper");
* let scissors = document.getElementById("scissors");

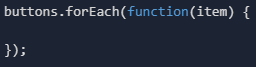
Also, we want to assign two variables as numbers and two variables as strings like so:

* let playerScore = 0;
* let computerScore = 0;
* let playerMove = “”;
* let computerMove = “”;

We also want to use something called “querySelectorAll()”. What this does is it looks for all instances of a specific HTML element. Let’s create a variable that uses this tool.



Furthermore, we want to use a “forEach” loop. If you remember, loops will repeat until a certain condition is met. The same thing applies to “forEach” loops. We will attach “buttons” to a forEach loop with an anonymous function.



From this, we will create an “addEventListener” that is attached to “item” and have the event listener respond every time we click it. To test that out, let’s console.log(item.id). This will print out the ID of the button that we click on.

Once that’s done, we should be seeing the button’s ID every time we click a button. HOWEVER, we want to find the specific button that is pressed. Let’s use if-statements to find both the id of the pressed button and the id of the move you can do (e.g. rock.id, paper.id, scissors.id).

After the if-statement, make a variable called:

let generateComputerMove = Math.floor(Math.random() \* 3);

* This does the following:
  + Math.random() gives us a random number between 0 and 1
  + Math.floor() rounds the number down
  + Math.random() \* 3 multiplies the random number by 3 so that we can get 0, 1, or 2

# Day 8 (Mar 3/4):

## **BUILDING A GAME (Rock Paper Scissors)**

### **Making a Functional Game with JavaScript**

Let’s first enable the computer to make a move like so:

| if(generateComputerMove == 0) {  computerSymbol = generateComputerMove;  computerMove = "The computer's move was rock!";  } else if(generateComputerMove == 1) {  computerSymbol = generateComputerMove;  computerMove = "The computer's move was paper!";  } else if(generateComputerMove == 2) {  computerSymbol = generateComputerMove;  computerMove = "The computer's move was scissors!";  } |
| --- |

At the top of the document, make two more variables called playerSymbol and computerSymbol and have them equal to -1

* let playerSymbol = -1;
* let computerSymbol = -1;

First, let’s go back to our first if-statement and set the playerSymbol to equal 0 when we click rock, playerSymbol = 1 when we click paper, and finally playerSymbol = 2 when we click scissors.

After that, we’ll go all the way until we get to the line underneath generateComputerMove. We have to make another if-statement there.

| if(playerSymbol < computerSymbol) {  if(playerSymbol == 0 && computerSymbol == 2) {  playerScore += 1;  } else {  computerScore += 1;  }  } else if(computerSymbol < playerSymbol) {  if(playerSymbol == 2 && computerSymbol == 0) {  computerScore += 1;  } else {  playerScore += 1;  }  } else {  //it was a tie  console.log("tie");  // alert("It was a tie!");  } |
| --- |

The if-statement contains the following:

* First, if-statement checking to see if the playerSymbol < computerSymbol meaning that the computer is winning
  + Within this, we must check if the player has played rock and the computer has played scissors. If this happened, we must increment the player’s score, otherwise, increment the computer’s score
* Second, else if-statement checking to see if the computerSymbol < playerSymbol meaning that the player is winning
  + Within this, we must check if the computer has played rock and the player has played scissors. If this happened, we must increment the computer’s score, otherwise, increment the player’s score
* Third, else statement to see if it is a tie
  + Display an alert to let the user know the match ended in a tie

CONGRATS YOUR GAME IS DONE

### **Adding Some Finishing Touches**

Now that we’ve finished our game, we should add some things to make it better than it currently is:

* HTML
  + Reset button
  + Two-player layout
  + Input a winner score limit
* CSS
  + Add some height to the body
  + Style the CSS for a mobile layout?
* JavaScript
  + End the game when the player/computer has won a certain amount of games

#### **HTML**

Let’s implement a reset button. Perhaps the game is not to your liking and you want to have another go at it. For this, we’ll have to put a new button called “Reset” with an ID of “reset” (all lowercase). It should look like this:

* <button id=”reset”>Reset</button>

We may also want to make a two player mode. For this, we’ll have to make another webpage with a link attached to it like so:

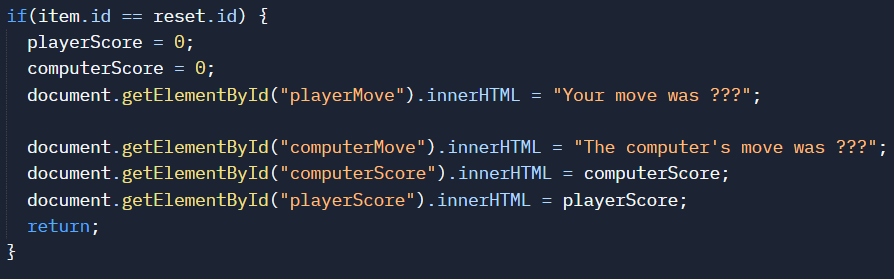
* <a href=”two\_players.html”>Two Player Mode</a>

Finally, we need something that tells us how much you need to win by. Let’s use a paragraph element like so:

* <p>Win By 3 Rounds</p>

#### **JavaScript**

If we want to reset the game, we’re going to have to set both scores to 0. We first need to get the ID of “reset” from the document and assign that to the variable called “reset”. The following code illustrates how we can do this:



# Day 9 (Mar 10/11):

## **BUILDING A GAME (Rock Paper Scissors)**

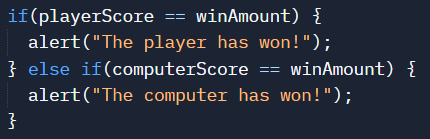
### **Adding Some Finishing Touches**

#### **JavaScript**

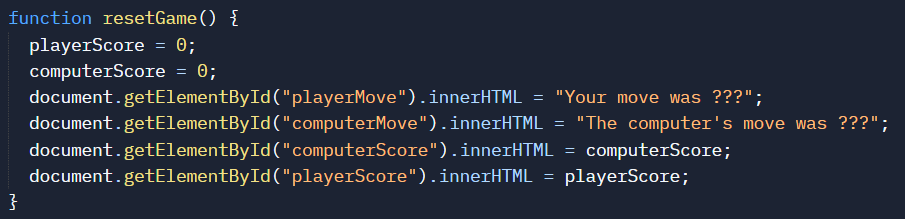
We need to end the game if the player or computer has won a certain amount of games. For this situation, we’ll need the following:

* A variable for the win amount
* Check if a user or computer has the same amount as the win amount variable
* Alert the user of who won the rounds
* Reset the game





This code will show us what to write for who has won the game! However, we also need to reset the game. Notice how we have already written the code to reset it but we don’t want to write that same code again at the bottom. Instead, we can write a function that holds the code to reset the game and we can call that function whenever we want.



Once this function is written, we need to figure out a way to **call** this function in our event listener code. Hint: go back to your other JavaScript workplace and look for a function called “paragraph()”

#### **HTML and CSS**

Go back into our HTML file and add another paragraph that let’s add a paragraph tag that says “Win by 3 Rounds”. If you want to (later), you can try to figure out a way to make that tag more responsive meaning that when you change the winAmount variable in JavaScript, that paragraph tag will also change in HTML.

Let’s also add “justify-content: center;” to our main tag and “text-align: center;” to our main’s paragraph tag

#### **Download to Desktop**

Repl.it has a download project feature but it doesn’t work. Instead, we’re going to have to do it in a more tedious way like making individual files for our code.

## **A Final Review of Everything**

### **HTML**

* Basics of HTML
  + Tags < >
    - When writing HTML, we use angle brackets to define the type of element we want to create
  + <!DOCTYPE html>
    - a.k.a. Document Type Definition
    - Always at the top of the page (like the very top)
    - Defines the type of HTML being used
    - There are 5 different types of HTML versions. The one we are using is HTML5, the most recent version
  + <html> </html>
    - Defines that we are using HTML and not any other web-based language
    - Everything we write must go in between these tags
  + <head> </head>
    - This is the head, or brain, of the document. Just like a human body, the head stores all the information such as your name, the things you like or dislike, and other objects like your age, the school you attend, and where you live.
    - <title> </title>
      * Found in the head, it is what that website page is called. Just like how you have a name, so does this specific page.
    - <meta charset=”utf-8”>
      * Also found in the head section, this charset lets us use almost any keyboard key that we want. Some computers may not take specific keys so it’s important to add an object that allows us to use almost any keyboard key to avoid any confusion
  + <body> </body>
    - Just like your body which has your torso and arms (hopefully), the website’s body also has content.
    - Tags you can use:
      * “p” tags
      * h1, h2, h3, h4, h5, h6
      * “a” tags
      * Tables
      * Articles and sections (I didn’t show you this)
      * Header, nav, main, footer
      * Images

### **CSS**

* Makes the website look nice through adding our own stylings
* Within an embedded or external style, we want two things:
  + The HTML tag (p, body, main, etc.) AND brackets ({ })
  + E.g. body { }
* Body { color: yellow; }
  + Body is the **selector**
  + Color is the **property**
  + Yellow is the **value**
* CSS elements:
  + Color: changes the color of the text
  + Background-color: changes the background color of the tag/object
  + Font-size: changes font size of a tag
  + Font-style: can change text style (italics, bold, etc.)
  + Font-family
  + Text decoration

#### **Classes and IDs**

* Imagine a school. You have the following:
  + Class(es) can be used for as many HTML elements as you want
  + id(s) are specific to one HTML element
  + Class
    - Denoted by .class
    - Can be used as many times as we want as long as we specify what element uses that tag
    - E.g. <h1 class=”top”> </h1> <p class=”top”> </p>
    - .top { font-size: 200%; color: green; }
  + Ids
    - Denoted by #id
    - Unlike classes, they can only be used for one element
    - E.g. <h2 id=”bottom”> </h2>
    - #bottom { background-color: blue; font-style: italic; }

### **JavaScript**

* Basics of JavaScript
  + Data types
    - We have a variety of different types of data in Javascript but the ones we will be focusing on are:
      * Boolean
      * Number
      * String
      * Undefined
  + Variables
    - Use “var” or “let” to define a variable (I’ll be using “let”)
    - Making a boolean
      * let isDaytime = true;
        + Let: variable
        + isDaytime: variable name
        + True: data type (in this case, boolean)

#### **If-Statements**

* We use if-statements to check if a condition is true or not
* We denote this via if(parameter) followed by curly brackets like so:
  + if(parameter { code }
* If our first statement isn’t true, we can have another condition by adding an “else if” which is followed after the curly brackets
  + else if(parameter)
* If nothing gets through our if and else if statements, then we can call an “else” condition
  + Else { }
  + An else statement is our default condition if nothing gets through our if statements
* Example:
  + if(hours == 1) { console.log(“Good morning”); } else if(hours == 2) { console.log(“Good afternoon”); } else if(hours == 3) { console.log(“Good night”); } else { console.log(“Good”); }

#### **Checking Multiple Conditions**

* When we want to check multiple conditions, we use “&&” or “||”
  + &&: both conditions must be satisfied for the operation to work
  + ||: one of the conditions must be satisfied for the operation to work
* Booleans
  + console.log(true && false); //returns false
  + console.log(true && true); //returns true
  + console.log(true || false); //return true
* Strings
  + console.log(“cat” && “dog”); //returns “dog”
  + console.log(“cat” || “dog”); //returns “cat”
* Numbers
  + console.log(5 > 0 && 2 < 0); //returns false
  + console.log(5 > 0 || 2 < 0); //returns true

#### **Functions**

* A block of code that is used for a specific task
* Need the following:
  + “Function” keyword
  + Function name: functionName
  + Parameter brackets: () (parenthesis)
  + Curly brackets for code { }
* A function must be called for it to run. This means that there must be a specific button or feature in your website that gets the function to compile its code
* Parameters in functions
  + These are the requirements we need for a function to run and parameters are written in the form of a variable name
  + E.g. function functionName(parameter);
  + Parameters can be used in numerous ways:
    - Solving a math problem using numbers
    - Getting someone’s name using strings
    - Answering questions using booleans
* Return
  + The “return” keyword can be used to end a function
  + Typically put at the end
  + Can return an operation, variable or a data type
    - E.g. return “word”, return true, return isDaytime;
  + Assigning a variable to a function can assign whatever return value that is in the function to the variable
    - Let currentTime = getTime(120);

#### **Calling a Function**

* Type the function name with the parameters (if it has no parameters, just have the parenthesis)
  + E.g. functionName() or functionName(parameter)
* We can call a function to the console by assigning a variable to it. If the function has any console.log code, then that code will be printed out

#### **Loops**

A loop is a coding mechanism in which you can repeat a task multiple times without having to rewrite it

##### **For-Loop**

* Start off with a “for” and have the parameters next to the loop like so: for(let i = 0; i < 5; i++)
* In the parameters, you need the following:
  + The starting condition: e.g. let i = 0;
  + The ending condition: e.g. i < 5;
  + The iteration count (how many times your starting condition will change for every loop): e.g. i++ which means “for every loop, the variable ‘i’ will increase by 1”
* Curly brackets are followed after every loop
  + Within the curly brackets is the code you want to write

##### **While Loop**

* Start with a “while” and have your parameter
  + Your parameter is a specific condition such that the loop will continue while your parameter is true
  + E.g. while(minutes < 60) { minutes += 1; }

##### **Do-While Loop**

* Basically the same thing as a while loop but its “do { } while(parameter)”

##### **Differences**

* for - loops through a block of code a number of times
* while - loops through a block of code while a specified condition is true
* do/while - also loops through a block of code while a specified condition is true
* Also, a while loop will check the parameters first and then execute the code if it is true while a do/while loop will run the code first and then check if the parameters are true

## **Closing Remarks**

First of all, thank you for learning HTML, CSS, and JavaScript with me. It was a pleasure to do this. If you’re interested in continuing with learning how to code, I would recommend you learn either **Python** or **Java** as well. In addition, we used Repl.it for this class but it honestly isn’t really that good. I recommend using [Visual Studio Code](https://code.visualstudio.com/) because it's a popular programming tool that is used at colleges, universities, and some jobs. However, it can be a bit hard to figure out how to use it so I’m not suggesting you learn it right away (but you will have to work with something like it in the future if you continue down this path). Here’s some links that may help:

* <https://code.visualstudio.com/learn/get-started/basics>
* <https://code.visualstudio.com/docs/introvideos/extend>

Otherwise, there is nothing else I would like to say. Again, thanks for joining my class and I hope you have a wonderful rest of your day.