**Web Development**

1. Brainstorming & Designing Tools
   1. Notion -> use for site design
   2. Pixelhunter.io 🡪 resize images for all social media
   3. Fontjoy 🡪 find font pairings
   4. Sketch.metademolab 🡪 animate drawings
   5. Trello -> Organizing things to do/ user story making
   6. Balsamiq.cloud/ tldraw -> Wire framing
   7. Alison.com
   8. Axiom.ai → browser automation
   9. Roadmap.sh → roadmaps
   10. Flowmapp → user stories
   11. Meshy → gradients
   12. Denigma → explains code
   13. Patterns.dev →  designs
   14. Checklist.design → best practices
   15. Lighthouse extension
   16. Colorhuddle.co 🡪 color schemes
   17. Loading.io 🡪 loading icons
   18. Videvo.net → video stock
   19. Letsenhance.io → image size
   20. Coolors.co 🡪 color schemes
   21. Greatlandingpagecopy.com
   22. Marketingexamples.com
   23. Genial.ly 🡪 spice up presentations
   24. Overapi.com 🡪 code cheatsheets
   25. App.haikei.app 🡪 custom shapes
   26. Page design inspo:
       1. Land-book.com
       2. Awwwards.com
       3. Siteinspire.com
   27. Milled.com 🡪 email marketing
   28. Brandcolors.net 🡪 colors for brands
   29. Icons
       1. Icons.js.org
       2. Visiwig.com
2. [CodinGame and CoderPad Tech Hiring Survey 2022 - CodinGame for Work](https://www.codingame.com/work/codingame-coderpad-tech-hiring-survey-2022/?utm_source=codingame&utm_medium=email&utm_campaign=survey-2022&utm_content=button)
3. [30 Must See Websites Built With Bootstrap – BootstrapBay](https://bootstrapbay.com/blog/built-with-bootstrap/)
4. Games to learn front-end dev
   1. JS Learning
      1. Warriorjs.com
      2. Codinggame.com
      3. Codecombat.com
   2. Cssbattle.dev
   3. Devchallenges.io
   4. 100dayscss.com
   5. Javascript30.com
   6. [Frontendmentor.io](https://www.frontendmentor.io/home)

**HTML**

**Notes**

1. Contains ‘tags’ and attributes and is known as a markup language
2. ! + tab ->gives you basic html template in vs code
3. ‘#’ harvard 🡪 to access id
   1. You can add an id to html elements
4. If you are using multiple attributes inside the style attribute, you must separate them with ‘;’
   1. style = “width: 125px; height: 225px;”
5. The <!DOCTYPE html> declaration at the top of the document tells the web browser that this is an HTML document.

The <html> element is the root element of the document and contains all of the other elements.

The <head> element contains information about the document, such as the title and any links to external stylesheets or scripts.

The <body> element contains the content of the document that is displayed in the web browser.

1. Classes can be defined by using ‘.’
   1. Ex 🡪 .centered { text-align:center}
      1. To use the class (in an html file) 🡪 <header class = “centered”>
      2. <header class = “centered large”> 🡪 To use more than one class at once
2. Basic html has

**<html >**

**<head>**

**<head>**

**<body>**

**<body>**

**<html>**

1. Hierarchy

|  |
| --- |
| <div >  <img src="Images/logo.png">  <ul>  <a href="" id="bar">Hello</a>  </ul> </div>  ## to reference the <a> you need to either assign it an id or reference it using ‘div ul a’ |

**HTML Attributes**

1. onload = “someFunction()”
2. onClick = “someFunction()”
3. value = “some string”
4. autofocus 🡪 allows cursor to automatically be in input box
5. autocomplete = “off” 🡪 self explanatory
6. data (an attribute)
   1. If you want to make your own attributes then you have to put it behind a ‘data-‘
      1. Ex. data-key= “65” 🡪 so that you're creating in attribute called k’key’
      2. You can cleverly associate tags together by giving them the same data attribute

Logo

Description automatically generated with low confidence

A picture containing text

Description automatically generated

**Selectors**

1. .intro, #Lastname ⇒ Selects both all elements with class="intro", and all elements with id="Lastname"
2. div p -> Selects all <p> elements that are inside a <div> element.
3. div > p -> Selects all <p> elements where the parent is a <div> element.
4. ul + p -> Selects all <p> elements that are next to <ul> elements.
5. p.myquote -> Selects all <p> elements with class="myquote".
6. [id] -> Selects all elements with an id attribute.
7. [id=my-Address] -> Selects all elements with an id attribute value equal to "my-Address"
8. [id$=ess] -> Selects all elements with an id attribute value ending with "ess"
9. [id|=my] -> Selects all elements with an id attribute value equal to "my" or starting with "my" followed by a hyphen (-)
10. [id^=L] -> Selects all elements with an id attribute value starting with the letter "L"
11. [title~=beautiful] -> Selects all elements with a title attribute value containing the word "beautiful"
12. [id\*=s] -> Selects all elements with an id attribute value containing the string "s"
13. :checked -> Selects all checked form elements.
14. :focus -> Selects the element that currently has focus.
15. p:first-child -> Selects all <p> elements that are the first child of their parent.
16. p::first-letter -> Selects the first letter of all <p> elements.
17. p::first-line -> Selects the first line of all <p> elements.

**Tags**

1. **<head>**
   1. <meta> 🡪 gives info about web page / does not have closing tag
   2. <meta charset=”UTF-8”>
   3. Has a closing tag
2. **<html lang=”en” >** 
   1. ‘lang’ 🡪An attribute that defines the language
      1. Adds meaning to that tag
3. **<title>** 🡪 Defines the name you see in the tab
   1. Goes inside the head tag
4. **<hr>** 🡪 Creates a line
5. **<body>** 🡪 The big rectangular part of the web page
   1. Attributes
      1. onload= “” 🡪 onload does as expected
   2. Has a closing tag
6. **<p>** 🡪 paragraph tag
   1. </p> to close
   2. Use to clearly separate out paragraphs rather than have a giant paragraph
   3. Has a closing tag
7. **<div>** 🡪 to make containers/divisions of content
   1. Has a closing tag
8. **<h1>** 🡪 biggest heading tag
   1. <h2>, <h3>, <h4> , <h5>, <h6>
   2. All different sizes
   3. Has a closing tag
9. **<ul>** 🡪 unordered list
   1. Has a closing tag
   2. Can put **<li>** inside
      1. 🡪 Unordered/ordered list items.
      2. You need an open and close tag for every item in the list
      3. Use attribute ‘list-style-type:’ to remove bullets or ordering or change the look
      4. Has a closing tag
10. **<ol>** 🡪 ordered list. It will number the things in the list for you
11. **<table>** 🡪 to enter tabular data
    1. **<tr> 🡪 (**table rows)
       1. inside <tr>, we have **<td>** 🡪Table data /column.
       2. Table data being inside table row means that the table data would be added horizontally
12. **<img alt = “harvard University”  src = “harvard.jpg”>**
    1. ‘alt’ 🡪 for those with vision issues
    2. The image must exist on IDE in a folder you can access or on remote desktop
    3. To use an img as a link, wrap it in a **<a>** tag.
       1. Apply the sizing to the <img> tag
    4. Attributes
       1. style
          1. width = “125px”
    5. Doesn't have a close tag
13. **<a href = “image.html” > Harvard </a>**
    1. The word being ‘hyperlinked’ is Harvard. It will send you to ‘image.html’
    2. To link a website, you must include the HTTP protocol/URL as the href
    3. Has a closing tag
14. **<nav>** 🡪 used for a big block of navigation links
    1. Has a closing tag
15. **<style>** 🡪 to change the color of links
    1. Used instead of a CSS file

|  |
| --- |
| <style>  h1 { color: #ff0000; text-decoration: none; }  → //this means change it for everything with this tag  h1:hover{ text-decoration: underline; }  → it is a pseudo selector  #Harvard { color: #000ff; }  → to change things for the tag whose id is Harvard  </style> |

1. **<form>**

|  |
| --- |
| <form action = "https://www.google.com/search" method = "get">  → Google doesn't need the get method, only requires the 'q' parameter  <input name = "q" type = "search"> <input type = "submit" value = "Search">  </form>  /\*Input tag does not need a closing tag //The action is where you want to lead users //the method is the HTTP verb you want to use(get,post), using "post" is more secure and hides searches) \*\ |

a. Inside a form, you can have an <input> with type = “submit” and that will function as a button.

1. **<footer>**
2. **<br>**  🡪 newline tag



* 1. No closing tag

1. **<button type = “button”>**
   1. You can have tags inside unlike with input buttons
   2. Attributes
      1. border-radius 🡪 to change shape
      2. type
         1. button
2. **<input>** 
   1. type
      1. submit
3. **<select>** 
   1. **<option>**

|  |
| --- |
| <select> <option value = 'black'> Black </option> <option value = 'red'>Red </option> </select> drop down menu |

**Extra**

1. To add a new font
   1. Go to google fonts and get the html link for the font,
   2. Add it to the head as a link
   3. Copy the ‘font-family: yada name;’ into your css file or <style> tag
2. To add icons
   1. For font awesome 5
      1. **<script** **src="https://kit.fontawesome.com/e4751e36ad.js"** **crossorigin="anonymous"></script>**
      2. **<i class="fa-solid fa-star"></i>**
         1. You don’t need anything else here
      3. To adjust icon size relative to container

Text, letter

Description automatically generated

* + 1. Anything you do to the parent that can be inherited will be inherited by the icon, for example, color changes

**Code**

1. Featured Category
   1. html

Text

Description automatically generated

* 1. css

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

**CSS - cascading /style**

**Notes**

1. Has ‘properties’ (key,value) pairs/ like attributes
2. The attributes of the parents become the attribute of the child if they can be inherited
3. To increase the size of a symbol or text inside a button or other thing use ‘font-size:50px;’
4. In CSS, a margin is the space around an element's border, while padding is the space between an element's border and the element's content.
   1. margin:0; 🡪 removes margin everywhere

Shape

Description automatically generated

1. **\*{**

Using the star means that everything you put in here will be applicable to all html elements

**}**

1. You can create a css file with classes that you later use in a html file and then just link them
   1. **<link href = “style.css” rel = “stylesheet” >**
2. Header/body are blocks by default
3. Images are inline
4. If the parent is set to ‘display:flex’, the heights/widths of the child items might have set values 🡪 you would need to use ‘flex-basis’ on the child item

**Pseudo Selectors**

1. hover
2. active 🡪 if the item is activated like clicked

**Common Attributes**

1. height
   1. 50vh 🡪 set the height 50% of the view port
   2. 50vw 🡪 set the width 50% of the viewport
2. order: number 🡪 can change the order of elements without changing the html
3. opacity 🡪 takes a number between 0 and 1
   1. 0 🡪 make invisible
   2. 1 🡪 completely visible
   3. To set the opacity of a background without making the text transparent as well, set opacity with rbg instead
      1. background: rgba(red, green, blue, alpha); 🡪 The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).
         1. background: rgba(0, 151, 19, 0.1); 🡪 sets the opacity to 10%
4. text-align
   1. center
5. font-size
   1. small
   2. 12px
6. display
   1. inline-block 🡪 to make horizontal
   2. block 🡪 takes up the whole width
   3. flex 🡪 allows for resizing and adjusting of the elements inside
7. Positions
   1. static 🡪 according to the normal flow of the page.
   2. relative 🡪 that if you change the top/left/right positions, it will change according to where it would be normally.
   3. fixed 🡪 the position is relative to the viewpoint, it will be at the same place even if the page is scrolled.
   4. absolute 🡪 relative to its nearest ancestor, so if you change the top/right/left it will change according to the position of the element its near (padding is not respected if both are absolute).
8. top 🡪 changes the starting point of a positioned item, but must be positioned for example ‘position: absolute;’
   1. 50px
9. border

Graphical user interface, text, application

Description automatically generated

* 1. dotted - Defines a dotted border
  2. dashed - Defines a dashed border
  3. solid - Defines a solid border
  4. double - Defines a double border
  5. groove - Defines a 3D grooved border. The effect depends on the border-color value
  6. ridge - Defines a 3D ridged border. The effect depends on the border-color value
  7. inset - Defines a 3D inset border. The effect depends on the border-color value
  8. outset - Defines a 3D outset border. The effect depends on the border-color value
  9. none - Defines no border
  10. hidden - Defines a hidden border
  11. The border-style property can have from one to four values (for the top border, right border, bottom border, and the left border).

Text, table

Description automatically generated

1. cursor

Graphical user interface, application

Description automatically generated 🡪 there’s more

1. Transition 🡪 transition: <property> <duration> <timing-function> <delay>;
   1. Text

      Description automatically generated
2. Transform:
3. box-shadow
4. Function ‘rgb(x,x,x)’ 🡪 to set your own color
5. <meta name =”viewport” content=width-width, initial-scale=1.0”>

**Flexbox**  🡪 layout for arranging things in a container. You have a main container that you assign the flex property to. Then you have flex items inside the container

1. Flexboxes have a main(x) and a cross(y) axis
2. If you add display: flex to a container:
   1. The items inside will be flex items and occupy equal portions of the space allotted to them. Unless you change the space they take up with flex: number (flex item attribute)
3. The flex property is a shorthand property for:
   1. flex-grow
   2. flex-shrink
   3. flex-basis 🡪 works as both to set width/height depending on the ‘flex-direction’
   4. Default values 🡪 0 1 auto
   5. ^ flex item attributes
4. Flexbox Attributes
   1. The flex-flow property is a shorthand property for:
      1. flex-direction
      2. flex-wrap
      3. default value: row nowrap
   2. justify-content: flex-start (end, center) 🡪 alignment
   3. justify-content: space-between 🡪 adds the margin in between the items. (flex container attribute)
   4. justify-content: space-around 🡪 adds margin in between and on the sides (flex container attribute)
   5. align-items: flex-start (center, end) 🡪 align on the ‘y’ axis. Will adjust the items based on height
   6. align-content: flex-start (end, center) 🡪 align on ‘x’ axis (flexbox attribute)/ doesn’t work in Chrome/ use justify-content instead
   7. flex-wrap: wrap (nowrap, wrapreverse) ⇒ whether or not item should go to the next line if screen is small
   8. flex-grow: number 🡪 specify how much space in a row you want an item to take up
   9. width: 20%; 🡪 applies to flex box; means flex items occupy 20% of the space of the parent container
5. Flex Item Attributes
   1. flex-grow 🡪 A number specifying how much the item will grow relative to the rest of the flexible items
   2. flex-shrink 🡪 A number specifying how much the item will shrink relative to the rest of the flexible items
   3. flex-basis 🡪 a width for your item
      1. 20% does the same thing as width: 20%
   4. align-self: flex-end… 🡪 apply directly to item
   5. flex: number 🡪 changes the amount of space an item takes up in the flex container

**Media Queries**  🡪 for size adjustments based on screen size or device, you can add media queries

1. Media can be specified with media types like **‘screen, print, speech, all’**
2. To use with a CSS file

 <link rel="stylesheet" media="screen and (min-width: 900px)" href="widescreen.css">

|  |
| --- |
| @media(min-width:768px){ } |
|  |

* 1. This means when the screen is at least 768px, whatever attributes you add in here will take place (this is different from device specific min and max. For that, you need max-device-width/min-device-width)
  2. The style changes as the screen itself changes

|  |
| --- |
| **@media screen and (max-width: 900px) and (min-width: 600px), (min-width: 1100px) {**   div.example {   font-size: 50px;     padding: 50px;    border: 8px solid black;     background: yellow;   } **}** |
|  |

* 1. ‘and’ functions as you would expect
  2. the comma ‘,’ functions as OR

**Responsive Website**

1. Units
   1. Absolute units ⇒ pixels(px), cm, mm 🡪 use for small details like border and shadow/these override users browser preferences
   2. Percentages ⇒ used for widths/relative to their parent except on heights 🡪 good to use for layouts height and width
   3. Relative Units ⇒ relative to font size/ or relative to viewport
2. Relative to font-size
   1. em 🡪 relative to the font size of parent if used on the font-size property, on all other properties, it will be relative to the font-size of the current element (1em, 1.5em, 2em)
      1. If you have a div with a font-size = 30px. And have an element inside with a font size of .5em 🡪 font-size of 0.5em equals to 0.5x30 = 15px
      2. Good to use on font size and margin/padding Like when you want to adjust padding or margin based on that elements font size
   2. rem 🡪 always relative to the root html element
      1. if the font size of the document is 16px, and an element has a font size of 2rem, that element’s font size would be 32px; 1rem is 16px if using the default html root font size
   3. vw/vh 🡪useful for big layouts
   4. ch 🡪 relative to the width of the number 0 of the current font. ‘ch’ is used to size the width of a paragraph. In general, you want a 45 to 70 character wide column for readability
      1. you use it like this:
   5. max-width: 40ch;
   6. this sets the width of the column to a maximum of 40 characters per line

**Specifics**

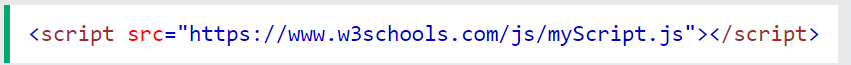
1. To center an image

|  |
| --- |
| display: block; margin-left: auto; margin-right: auto; |

**JavaScript**

**Notes**

1. To run JS in a browser, you need an HTML file but with a node.js extension you can run it on an IDE like Vs code
2. There are no ‘ ; ’
3. Can have a function as an argument
4. Window object
   1. window object is the global scope object, that means that variables, properties, and methods by default belong to the window object. This also means that specifying the window keyword is optional
5. Client side programming
6. Uses <script> tags inside HTML or use in a JS file
   1. **<script src = script.js> </script>** 🡪 to link to a separate file
   2. The script will behave as if it was located exactly where the <script> tag is located.
   3. An external script can be referenced in 3 different ways:
      1. With a full URL (a full web address)



* + 1. With a file path (like /js/)
    2. Without any path

1. ‘ + ’ 🡪 for concatenation
2. To have functions run on their own, set an interval
   1. setInterval(count, 1000); 🡪 says to run count every second/1000 milliseconds
3. To get the value the user types inside an input field you need to use ‘.value’
4. You can add properties to almost anything
5. window.print()  🡪 method in the browser to print the content of the current window.
6. ‘{}’ 🡪 creates an object
   1. Object properties are written as name value pairs separated by commas

A picture containing graphical user interface

Description automatically generated

* 1. var a = {}

a.b = ‘hello’

🡪 by adding a value, you are saying that the object ‘a’ has a property called ‘b’ whose value is ‘hello’

1. [] 🡪 creates an array
   1. var a = []
2. A function is an object
3. To avoid creating global variables you should put your variable declarations inside of a function and then call that function to use the variable

Text

Description automatically generated

1. In JavaScript, variables declared with var are “hoisted” to the top of the function they’re declared in. This code:

Graphical user interface, text

Description automatically generated with medium confidence Text, letter

Description automatically generated

1. ‘this’ 🡪 In other languages, this means the instance of the class that you’re looking at, but in JavaScript it means the object that your function got called on

Graphical user interface, text

Description automatically generated

foo method puts a value on a‘s message property

1. Evaluates from left to right

Graphical user interface, text, application

Description automatically generated Graphical user interface, text, application

Description automatically generated

1. Automatic type conversion
2. Strings can use single or double quotes
   1. to include quotes inside of a string, the quotes on the end side cannot match the surrounding string



1. Literals 🡪 Fixed values
   1. Numbers, strings
2. Variables 🡪 variable values
3. Comments 🡪 // or between /\* and \*/
4. JavaScript uses the Unicode character set
5. Block scope 🡪 Variables declared inside a { } block cannot be accessed from outside the block

A picture containing graphical user interface

Description automatically generated

1. Function scope 🡪

**Operators**

1. ( + - \* / )
   1. \*\* 🡪 raises the first operand to the power of the second operand
2. ‘typeof’ 🡪 to find the type of a JavaScript variable/expression

A picture containing graphical user interface

Description automatically generated

1. ‘instanceof’ 🡪 Returns true if an object is an instance of an object type
2. == 🡪 equal to operator
3. Comparison operators

Graphical user interface

Description automatically generated with medium confidence

1. Logical Operators

Table

Description automatically generated with medium confidence

**Syntax**

1. Ending statements with semicolon is not required, but highly recommended
   1. When separated by semicolons, multiple statements on one line are allowed

A picture containing logo

Description automatically generated

1. Variable naming
   1. Upper Camel Case (Pascal Case):
      1. FirstName, LastName, MasterCard, InterCity.
   2. Lower Camel Case:
      1. firstName, lastName, masterCard, interCity.

**Attribute Selectors**

1. Similar to attribute selectors in CSS. 🡪 they allow you to be even more specific in your queries
   1. i.e if you want to find an element with a specific attribute

Text

Description automatically generated with low confidence

This says, find me an audio element with an attribute, data-key whose value is 65.

A picture containing text

Description automatically generated

Same thing as above except now we are using **template strings** to make the function return the audio element associated with any key that is pressed

* + 1. notice that we had to use backticks here and put quotes around the template string

**Best Practice**

1. If a JavaScript statement does not fit on one line, the best place to break it is after an operator

Text

Description automatically generated with low confidence

1. Underscores 🡪 a convention among professional programmers is to use it as an alias for "private (hidden)" variables
2. $ 🡪 but professional programmers often use it as an alias for the main function in a JavaScript library

**Variables/Identifiers**

1. Hyphens are not allowed
2. The let and const keywords were added to JavaScript in 2015
3. var
   1. variables can be called before definition (var are **hoisted** to the top and can be initialized at any time) 🡪 the type will be ‘unidentified’

A picture containing text

Description automatically generated

* 1. If you re-declare a JavaScript variable declared with var, it will not lose its value.
     1. The variable carName will still have the value "Volvo" after the execution of these statements

A picture containing chart

Description automatically generated

* 1. If you want your code to run in older browser, you must use var
  2. Variables declared with the var keyword can NOT have block scope

Chart

Description automatically generated with low confidence

* 1. Redeclaring a variable inside a block will also redeclare the variable outside the block

Text

Description automatically generated 🡪 Redeclaring a variable using the let keyword can solve this problem

1. let
   1. can’t be called before definition (ReferenceError)

A picture containing text

Description automatically generated

* 1. You cannot re-declare a variable declared with let or const

But Text

Description automatically generated A picture containing graphical user interface

Description automatically generated

* 1. Variables defined with let have Block Scope

1. const 🡪 the value of these variables can’t be changed (static)
   1. You CAN NOT
      1. Reassign a constant value
      2. Reassign a constant array
      3. Reassign a constant object
   2. But you CAN:
      1. Change the elements of constant array
      2. Change the properties of constant object
   3. Cannot be used before declared 🡪 hoisted to the top but not initialized

**Output**

1. JS ‘displays or prints to screen in the following ways
   1. Writing into an HTML element, using innerHTML.
      1. document.getElementById(‘id’).innerHTML = …..;
   2. Writing into the HTML output using document.write()
      1. Using document.write() after an HTML document is loaded, will **delete all existing HTML**



* 1. Writing into an alert box, using window.alert()
  2. Writing into the browser console, using console.log()

1. Clearing the window
   1. Document.write() 🡪 but this removes literally everything including the <html>
   2. Document.body.innerHTML= “”;

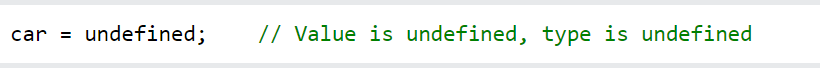
**Data Types**

1. Primitives 🡪 they don't have properties
   1. String
      1. Different from the object ‘new String(‘hi’)
   2. Number
      1. The only number type JavaScript has for example it does not have floats ints or longs
   3. Boolean 🡪 true or false
2. Dynamically typed language 🡪 (also called loosely typed) scripting language. That is, in JavaScript variables can receive different data types over time

Graphical user interface, application, chat or text message

Description automatically generated

1. const 🡪 the value of these variables can’t be changed (static)
2. undefined 🡪  a variable without a value, has the value undefined. The type is also undefined
   1. Any variable can be emptied, by setting the value to undefined. The type will also be undefined



* 1. It's different from an empty value, for example, in the expression ‘let car = “ “;’ 🡪 The value is " ", the typeof is "string"

**Loops**

1. ‘for in’
   1. Not like a ‘for each’ loop
   2. This loop gives you the *keys* by which other values can be found. You can get the values in other ways

Chart, scatter chart

Description automatically generated

* 1. Best used to iterate over key value pairs or properties of an object

1. ‘for of’ 🡪 iterate over values

**Important**

1. Unlike Java which does whole number division, 5/2 will give 2, since js doesn’t have float/ints, 5/2 in JS will produce 2.5
2. If statements just like java
3. While / for loops same as java(except use ‘let’ rather than ‘int’)
4. Concatenation 🡪 alert("Hello " + name);
5. Using JS functions in HTML:

|  |
| --- |
| <form onsubmit = "greet(); return false" <input id ="name" type ="text"> <input type = "submit"> </form> |

|  |
| --- |
| function greet() { let name = document.querySelector('#name').value  alert('hello, body') } |

**Vanilla Functions**

1. .includes() 🡪 can take either 2 args, the first being the searchElement, and the second being the index, or just one arg being the searchElement
   1. Called on:
      1. Array
      2. String (case sensitive)
   2. Returns a Boolean
   3. Ex. \_string.includes(“Firefox);

**HTML + JS**

**Methods**

1. getElementById()
   1. .src(‘Ex, the img file name’) 🡪 (or other HTML attributes)
   2. .style ( & accompanying css attributes, ‘.style.display, etc)
   3. .innerHTML = …….;
   4. You can use JavaScript to dictate the inner HTML of an element like so

A picture containing text

Description automatically generated

1. .addEventListener()
   1. Can add to:
      1. Window
      2. Document
      3. Textarea
   2. Events 🡪 must go inside quotes
      1. ‘keydown’
   3. Can access specifics about an event using ‘e’
      1. e 🡪 an object full of data that describes what happened

A picture containing text

Description automatically generated

Here ‘e’ will turn us about the key that was pressed

* + 1. e.keyCode() 🡪 will return the number associated with the key that was pressed

1. .querySelector() 🡪 to find one specific element 🡪 returns that element
2. .querySelectorAll() 🡪 to find all elements that match your query
3. .createElement(“div”)
   1. Called on:
      1. document
   2. the element you create must get appended to another element that already exist

**Graphical user interface, text, application

Description automatically generated with medium confidence**

* + 1. You can also append elements to the element you create
       1. You might want to add text inside by appending an element created with .createTextNode()



* 1. you can set attributes for it
     1. ****
  2. To append the new element to a blank html page
     1. Graphical user interface, text

        Description automatically generated with medium confidence

**DOM Manipulation**

1. Adding Event Listener

|  |
| --- |
| document.querySelector("form").addEventListener("thing to listen for", thing to call but leave out the ()) |

* 1. If ‘form’ does not exist til later:

|  |
| --- |
| function greet() {  let name = document.querySelector('#name').value  alert('hello, body')}  Function listen(){  document.querySelector('form').addEventListener('submit',greet);  } document.addEventListener("DOMContentLoaded,listen); |

|  |
| --- |
| document.querySelector('button').onClick = count; |

* 1. → Setting the value of the button's onclick function to count;  
     → Only when the button is clicked on should the function be called

|  |
| --- |
| document.addEventListener('DOMContentLoaded', function(){document.querySelector('button').onClick = count;    } ); |

* + 1. Can add event listener to any HTML element
    2. here we're adding it to the entire document
    3. Takes two arguments
       1. First 🡪 what event are you listening for 🡪 i.e ‘click or ‘scroll’ or ‘DOMContentLoaded’
       2. Second 🡪 what function should run when the event happens
          1. You can write a function name or you can define the function as the second argument
  1. document.querySelectorAll() 🡪 gets all the elements that match your query and puts them in an array 🡪 you can index into it
     1. i.e document.querySelectorAll(‘button’)
     2. To loop over the array:
        1. document.querySelectorAll(‘button’).forEach(function(button){button.onclick = function () {document.querySelector(‘#hello’).style.color = button.dataset.color;}}

1. To create an element:

|  |
| --- |
| const li = document.createElement('Li');  li.innerHTML = tasks(an unordered list);  document.querySelector('#tasks).append(li)  document.querySelector('#tasks).value = ''; → to clear input field |

1. To disable a submit button until text is written in input field:

|  |
| --- |
| document.querySelector('#submitButton).disabled = true;  document.querySelector('#input').onkeyup = () => { document.querySelector('#submitButton').disabled = false;  } |

1. Local storage 🡪 stores values inside the user’s web browser so that for example, if we have a counter function, it won’t restart each time we open the page

|  |
| --- |
| localStorage.getItem(key); localStorage.setItem(key,value);   if(!localStorage.getItem('counter')){            localStorage.setItem('counter',0);}  function count() {     let counter = localStorage.getItem('counter');     counter++;     localStorage.setItem('counter', counter); } |

1. To change the inner HTML of elements:

|  |
| --- |
| document.querySelector('h1').innerHTML = 'Goodbye'; à allow you to look through and HTML page and select an element   → Will only return the first h1 element it finds if(document.querySelector('h1').innerHTML === 'Hello!'){ document.querySelector('h1').innerHTML = 'Goodbye';}   → the === checks for strict equality; the values are equal and their types are the same thing;  i.e if one is a string, the other must also be a string |

1. Toggle Menu

|  |
| --- |
| <ul id= "MenuItems"> <... onClick= "menutoggle()">  <style> overflow:hidden; transition: maxHeight .5s;  <style>  <script>      var MenuItems = document.getElementById("MenuItems);      MenuItems.style.maxHeight = "0px";  function menutoggle(){    if(MenuItems.style.maxHeight =="0px"){       MenuItems.style.maxHeight = "200px"; } else{    MenuItems.style.maxHeight = "0px"; }   } </script> |

1. JS to change CSS
   1. i.e document.querySelector(‘#hello’).style.color = ‘red’;
   2. <button data-color=’red’></button>

document.querySelectorAll(‘button’).forEach(function(button){button.onclick = function () {document.querySelector(‘#hello’).style.color = button.dataset.color;}} 🡪 dataset lets you get access to data-color or whatever data element you want

|  |
| --- |
| document.addEventListener('DOMContentLoaded', function() { document.querySelector('select').onchange = function () { document.querySelector('#hello').style.color = this.value;} } |

🡪 keyword ‘this’ refers to the thing on which the event handler was called on

1. Formated String
   1. alert(`Count is now ${counter}`); 🡪 to include variables inside a string
      1. ` 🡪 back tick
      2. ${variable}
2. Dev Tools 🡪 console shows JS related things
   1. Shows errors
   2. Can also write JS code to see what a code will do
      1. Can set variables
3. Arrow notation
   1. Rather than writing function(){} you can do () => {}
      1. Or function(button){} 🡪 button => {}
4. JS object 🡪 similar to python dict
   1. Let person= { first: ‘Harry’, last: ‘Potter’};
   2. API
      1. Well defined way for services on the internet to communicate with each other
      2. Often written in JSON 🡪 JS Object Notation
   3. AJAX 🡪 asynchronous JS 🡪 even after a page has loaded, using JS we can make additional web requests to ask for additional information
      1. fetch() 🡪 makes a web request/ gets back an HTTP response
         1. i.e fetch(exchangerates api URl)
      2. after fectch() 🡪 include .then() 🡪 this tells the computer what to do after the info requested is received
         1. .then(response => {return response.json()} 🡪 return the response in a json format
         2. .then(data => {console.log(data);}
   4. A screenshot of a computer screen

      Description automatically generated with medium confidence