HTML & CSS

Basics:

- ☐ HTML full form, use, markup language vs programming language
- ☐ Introducing VS code
- ☐ Creating files and folders in VScode with extensions (.html, .css, .js, etc.)
- ☐ Head, body, title
- ☐ Tags(opening & closing)
- ☐ The folder structure in HTML / vs code
    - ☐ Assets (folder)
        - ☐ Images (folder)
            - ☐ SVG (folder)
            - ☐ Png (folder)
        - ☐ CSS (folder)
            - ☐ Style.css (file)
        - ☐ JS (folder)
        - ☐ Fonts (folder)
        - ☐ Sass (folder)
    - ☐ Index.html (file)

Tags:

- ☐ Basic Markup <!DOCTYPE>
- ☐ Syntax of elements and types.
- ☐
,<hr>, <del>,<b>, etc.
- ☐ entity/symbol
- ☐ Img(self closing tag)
- ☐ Knowledge about Elements and attributes (properties and values etc.)
- ☐ Links (anchor tag),
    - ☐ href=” ”
    - ☐ Target=” ”
    - ☐ image inside <a>
- ☐ List , <ul>, <ol> <ul style=”list-style-type:disc; > (in CSS)
- ☐ CSS Introduction (inline, internal, external)

- ☐ CSS Syntax
- ☐ CSS property and values i.e. Font size, color, background color, Align, etc.
- ☐ Padding, margin, etc.
- ☐ Border, border-radius
- ☐ CSS Id & Class
- ☐ Table (<th>, <tr>, <td>) border, border-collapse.
- ☐ Form (label+ input and related attributes), Select Tag
- ☐ Audio, video, iframe
- ☐ Box model
- ☐ Box-sizing: border-box; etc.
- ☐ CSS selectors
- ☐ Import Fonts
- ☐ Styling Tables
- ☐ Border
- ☐ Hover
- ☐ Outline
- ☐ Display
    - ☐ Block
    - ☐ Inline
    - ☐ Inline-block
    - ☐ Flex
- ☐ Positioning
    - ☐ static
    - ☐ Absolute
    - ☐ Relative
    - ☐ Fixed
    - ☐ Sticky
- ☐ Stacking order(z- Index)
- ☐ Pseudo selectors, classes (first child, last child, nth-child, etc)
- ☐ Pseudo-elements (First-line, first-letter, marker, selection, After and Before)
- ☐ Media Query/ min-width & max-width
    - ☐ Breakpoints
- ☐ Advance Border Radius

- ☐ Styling Backgrounds (bg-repeat, bg-size, position, etc.)
    - ☐ Background-image
        - ☐ Background Repeat
        - ☐ Background position
        - ☐ Background size
        - ☐ linear Gradients
    - ☐ Multiple images in the background
- ☐ Transform (2D & 3D)
- ☐ Transition
- ☐ CSS Animation
    - ☐ @keyframe
        - ☐ From and to
        - ☐ 0% to 100% etc.
- ☐ root in CSS
- ☐ Overflow properties
- ☐ Styling Scroll Bar
- ☐ Timeline
- ☐ Slick slider

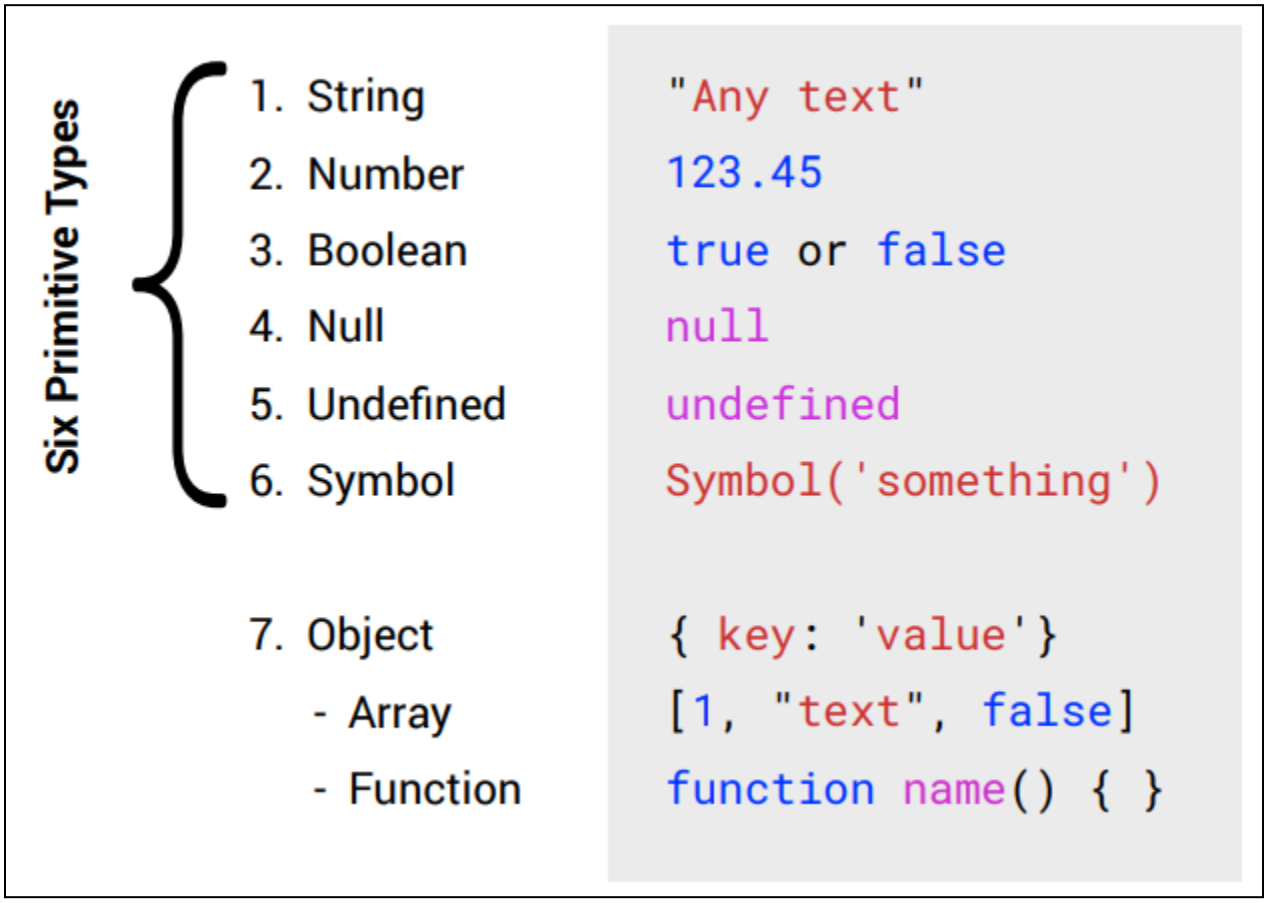
BootStrap

- ☐ Introduction
- ☐ Setup
- ☐ Bootstrap classes
- ☐ Breakpoints
- ☐ Container
- ☐ Grid system
- ☐ Navbar
- ☐ Modal
- ☐ Carousel
- ☐ Bootstrap Grid System
- ☐ Responsive Web Design
- ☐ Bootstrap Navbar
- ☐ Bootstrap Breadcrumbs
- ☐ Bootstrap Buttons

- ☐ Bootstrap Tables
- ☐ Bootstrap Forms
- ☐ Dropdowns
- ☐ Tabs
- ☐ Pagination
- ☐ Progress Bars
- ☐ Typography
- ☐ Tooltips
- ☐ Alerts
- ☐ Accordion
- ☐ ScrollSpy

Java Script

- ☐ What is JS and how to use it? History of JS?
- ☐ Data types



- ☐ Basic Vocabulary

Variable
A named reference to a value is a variable.

Operator
Operators are reserved-words that perform action on values and variables.
Examples: `+` `-` `=` `*` `in` `===` `typeof` `!=` ...

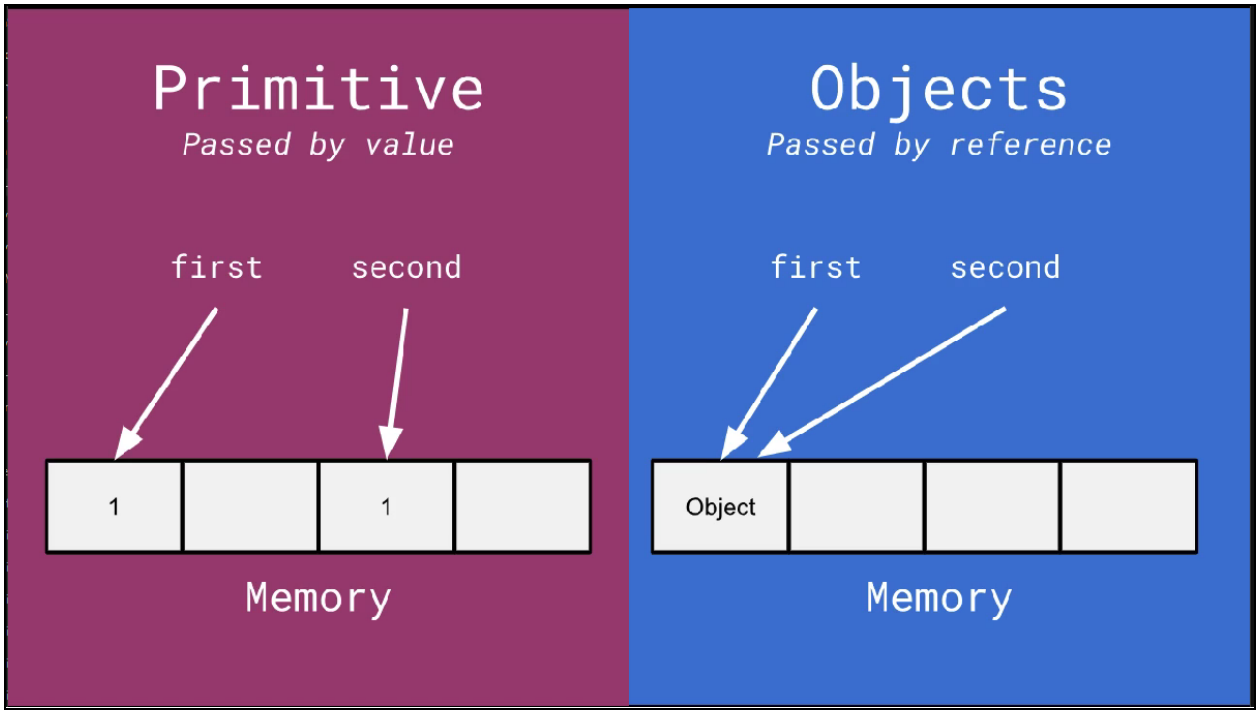
```
var a = 7 + "2";
```

Statement
A group of words, numbers and operators that **do a task** is a statement.

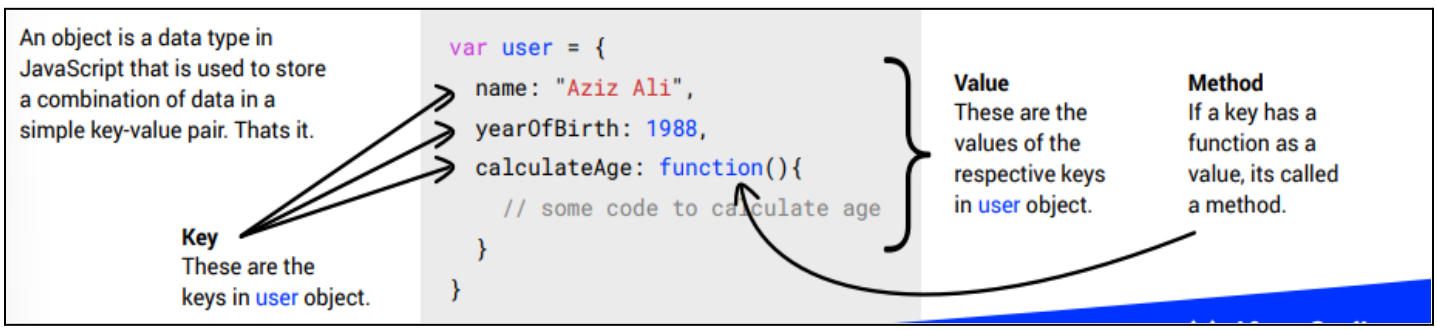
Keyword / reserved word
Any word that is part of the vocabulary of the programming language is called a keyword (a.k.a reserved word).
Examples: `var` `=` `+` `if` `for`...

Expression
A reference, value or a group of reference(s) and value(s) combined with operator(s), **which result in a single value.**

- ☐ Variable
- ☐ Operator
- ☐ Statement
- ☐ Expression
- ☐ Keywords etc.
- ☐ Passed by value and Passed by reference



- ☐ Object



- ☐ Array
- ☐ JS Functions (IIFE)

A function is simply a bunch of code bundled in a section. This bunch of code ONLY runs when the function is called. Functions allow for organizing code into sections and code reusability.

Using a function has ONLY two parts. (1) Declaring/defining a function, and (2) using/running a function.

Parameters / Arguments (optional)
A function can optionally take parameters (a.k.a arguments). The function can then use this information within the code it has.

Name of function
Thats it, its just a name you give to your function. Tip: Make your function names descriptive to what the function does.

```
// Function declaration / Function statement
function someName(param1, param2){

    // bunch of code as needed...
    var a = param1 + "love" +  param2;
    return a;
}

// Invoke (run / call) a function
someName("Me", "You")
```

Code block
Any code within the curly braces { ... } is called a "block of code", "code block" or simply "block". This concept is not just limited to functions. "if statements", "for loops" and other statements use code blocks as well.

Return (optional)
A function can optionally spit-out or "return" a value once its invoked. Once a function returns, no further lines of code within the function run.

Invoke a function
Invoking, calling or running a function all mean the same thing. When we write the function name, in this case someName, followed by the brackets symbol () like this someName(), the code inside the function gets executed.

Passing parameter(s) to a function (optional)
At the time of invoking a function, parameter(s) may be passed to the function code.

- ☐ JS Hoisting

Variable Declaration

The creation of the variable.

```
var a;
```

Variable Initialization

The initial assignment of value to a variable.

```
a = 12;
```

Variable Assignment

Assigning value to a variable.

```
a = "me";
```

Hoisting
Variables are declared at the top of the function automatically and initialized at the time they are run.

```
console.log(a);
var a = "me";
```

Scope
The limits in which a variable exists.

Global scope
The outermost scope is called the Global scope.

Functional scope
Any variables inside a function.

Lexical Environment (Lexical scope)
The physical location (scope) where a variable or function is declared is its lexical environment (lexical scope).

Rule:
(1) Variables in the outer scope can be accessed in a nested scope; But variables inside a nested scope CANNOT be accessed by the outer scope. (a.k.a private variables.)
(2) Variables are picked up from the lexical environment.

Scope chain

The nested hierarchy of scope is called the scope chain. The JS engine looks for variables in the scope chain upwards (its ancestors, until found).

```
var a = "global";

function first(){
    var a = "fresh";

    function second(){
        console.log(a);
    }
}
```

- ☐ Code formatting
- ☐ Closure and Lexical Scope
- ☐ JS Object Methods
- ☐ JS string Methods
- ☐ Array methods

Auto Inherited Properties:
When you create a value in JavaScript, certain properties are automatically inherited by this value. This magic happens because every type has a constructor with a special property called a prototype. All methods on the prototype gets automatically inherited by the new value created for that type. Take a look at some of of these methods on the right.

```
const thing = "some text";
```

String

Google 'Mozilla String' to find the docs

```
.concat()
.charAt()
.indexOf()
.startsWith()
.endsWith()
.split()
.slice()
```

```
const num = 123.45;
```

Number

Google 'Mozilla Number' to find the docs

```
.toFixed()
.toPrecision()
.toString()
```

Boolean

Google 'Mozilla Boolean' to find the docs

```
.toString()
```

Built-in Objects:

JavaScript gives us a ton of useful built-in objects to make our lives easier. The Date and Math objects are very useful on a regular basis. Take a look at some of their features on the right.

Math

Google 'Mozilla Math' to find the docs

```
Math.pow(2, 3)        // 8
Math.sqrt(16)         // 4
Math.min(7, 8, 6)     // 6
Math.max(7, 8, 6)     // 8
Math.floor(123.45)    // 123
Math.ceil(123.45)     // 124
Math.round(123.45)    // 123
Math.random()         // 0.45..
```

Date

Google 'Mozilla Date' to find the docs

```
const d = new Date('9/17/1988');
d.getDay()
d.getFullYear()
d.getMonth()

Date.now()
Milliseconds since Jan 1, 1970
```

- ☐ Operators

Operators are reserved words that perform an action on values and variables.

Arithmetic

.. + .. Add
.. - .. Subtract
.. * .. Multiply
.. / .. Divide
.. % .. Remainder
.. ** .. Exponential

Assignment

.. = .. Assign value
.. += .. Add then assign
.. -= .. Subtract then assign
.. *= .. Multiply then assign

Relational / Comparison

.. >= .. Greater than or equal to
.. <= .. Less than or equal to
.. != .. Not equal after coercion
.. !== .. Not equal

Increment / Decrement

..++ Postfix increment
..-- Postfix decrement

++.. Prefix increment
--.. Prefix increment

Logical

.. || .. Or

.. && .. And

Equality

.. === .. Equality

.. == .. Equality with coercion

Conversion

+ .. Convert to number

- .. Convert to number then negate it

! .. Convert to boolean then inverse it

- [ ] JS Coercion

When trying to compare different "types", the JavaScript engine attempts to convert one type into another so it can compare the two values

Type coercion priority order:

1. String
2. Number
3. Boolean

Coercion in action

Does this make sense?

```
2 + "7";   // "27"
true - 5   // -4
```

Conditional statements allow our program to run specific code only if certain conditions are met.

If-else Statement: Run certain code, "if" a condition is met. If the condition is not met, the code in the "else" block is run (if available.)

```
if (a > 0) {
  // run this code
} else if (a < 0) {
  // run this code
} else {
  // run this code
}
```

Switch Statement: Takes a single expression, and runs the code of the "case" where the expression matches. The "break" keyword is used to end the switch statement.

```
switch (expression) {
  case choice1:
    // run this code
    break;

  case choice1:
    // run this code
    break;

  default:
    // run this code
}
```

Ternary Operator: A ternary operator returns the first value if the expression is truthy, or else returns the second value.

```
(expression)? ifTrue: ifFalse;
```

- ☐ Truthy/ Falsy values:

There are certain values in JavaScript that return true when coerced into a boolean. Such values are called truthy values. On the other hand, there are certain values that return false when coerced to boolean. These values are known as falsy values.

Truthy Values	Falsy Values
true	false
"text"	""
72	0
-72	-0
Infinity	NaN
-Infinity	null
{}	undefined
[]	

- ☐ Loops
- ☐ Ways to create a variable
- ☐ SetInterval and clear Intervals
- ☐ SetTimeout and clear Timeout
- ☐ DOM

What is a "Node"?
(in the context of DOM)

Node: Every item in the DOM tree is called a node. There are two types of node - A text node, and an element node:

Text Node: Node that has text.
Element Node: Node that has an element.

Child Node: A node which is a child of another node.
Parent Node: A node which has one or more child.
Descendent Node: A node which is nested deep in the tree.
Sibling Node: A node that share the same parent node.

Query/Get Elements

```
// Preferred way:
document.querySelector('css-selectors')
document.querySelectorAll('css-selectors', ...)

// Old ways, and still work:
document.getElementsByTagName('element-name')
document.getElementsByClassName('class-name')
document.getElementById('id')
```

Create / clone Element

```
document.createElement('div')
document.createTextNode('some text here')
node.cloneNode()
node.textContent = 'some text here'
```

Add node to document

```
parentNode.appendChild(nodeToAdd)
parentNode.insertBefore(nodeToAdd, childNode)
```

Modify Element

```
node.style.color = 'red'
node.style.padding = '10px',
node.style.fontSize = '200%'

node.setAttribute('attr-name', 'attr-value')
node.removeAttribute('attr-name')
```

Get and Modify Element Class

```
node.classList
node.classList.add('class-name', ...)
node.classList.remove('class-name', ...)
node.classList.toggle('class-name')
node.classList.contains('class-name')
node.classList.replace('old', 'new')
```

Remove Node

```
parentNode.removeChild(nodeToRemove)
// Hack to remove self
nodeToRemove.parentNode.removeChild(nodeToRemove)
```

Get Element Details

```
node.nextSibling
node.firstChild
node.lastChild
node.parentNode
node.childNodes
node.children
```

Events

```
node.addEventListener('event-name', callback-function)
node.removeEventListener('event-name', callback-function)
```

- ☐ CRUD
- ☐ Promise
- ☐ “this” keyword
- ☐ Constructor

React-JS

- ☐ What is React JS
- ☐ React folder structure
- ☐ React UI
- ☐ React Components
- ☐ JS and JSX
- ☐ React Bootstrap
- ☐ React slick slider
- ☐ React Routing
- ☐ Props
- ☐ React Hooks
    - ☐ UseState
    - ☐ UseRef
    - ☐ UseEffect
    - ☐ UseContext(extra)
- ☐ Form Submission
- ☐ API fetch(extra)