HTML

- hypertext markup language
- used to designing the web ui
- invented by sir Tim Bernerse Lee on a NextStep Workstation
- **markup language**
    - which is made up of
        - tag
            - word enclosed by < and >
            - also known as element
            - e.g.
                - : paragraph
                - <table>: table
            - used as an instruction to perform some operation
            - types
                - opening
                    - used to start an instruction
                    - also known as starting tag
                    - e.g.
                        -
                - closing
                    - used to end the instruction
                    - also known as ending tag
                    - e.g.
                        -
                - empty
                    - tag without having any data
                    - e.g.
                        -
                    - shorthand
                        -
                - root
                    - which starts and ends the document
                    - e.g.
                        - in html, the <html> is the root tag`
            - attribute
                - more information about the tag

                - attributes are optional
                - if used then attribute must be used in the name = value format
                    - e.g.
                        - <input type="text" >
                        - where
                            - input is a tag
                            - type is an attribute
                            - text is the value for attribute type
                - for html5 attributes
                    - if attribute name and value is same then one can use the shorthand attribute
                    - e.g.
                        - <input required="required">
                        - <input required>
                - every html tag has following attributes
                    - **id**: used to identify the tag uniquely
                    - **name**:
                        - used to add the name for a tag
                        - used while submitting the form
                    - **style**:
                        - used to inline style to a tag
                    - **class**:
                        - used to add css class to a tag

            - data

                - also known as content
                - information enclosed by opening and closing tags
                - e.g.
                    - This is my paragraph
                    - where
                        - : starting tag
                        - this is my paragraph: data
                        - : closing tag

    - is not a programming language

dev environment

- editor: Visual Studio Code
    - extensions
        - html snippets (https://marketplace.visualstudio.com/items?itemName=abusaidm.html-snippets)
        - html boilerplate (https://marketplace.visualstudio.com/items?itemName=sidthesloth.html5-boilerplate)
        -

html tags

- comment
    - ignored while rendering the html
    - used to write single or multi line comments
    - e.g.
        - <!-- this is the head section -->
        - <!--
          this is the comment 1
          this is the comment 2
          this is the comment 3
          this is the comment 4
          -->
- **head section**
    - contains the meta data of the page
    - the tags are not meant to display any output
    - tags
        - **title**: used to change the tab title
        - **style**: used to add internal CSS code
        - **script**: used to add javascript code
        - **meta**: used to add meta (extra) information
        - **link**: used to link external documents
        - **base**: used for setting the base url for the website
- **body section**
    - **text tags**
        - heading tags
            - used to add headings
            - block tags
            - types
                - h1:
                    - `<h1>this is heading 1 </h1>`
                - h2
                    - `<h2>this is heading 2 </h2>`
                - h3
                    - `<h3>this is heading 3 </h3>`
                - h4
                    - `<h4>this is heading 4 </h4>`
                - h5

            - `<h5>this is heading 5 </h5>`
        - h6
            - `<h6>this is heading 6 </h6>`
    - paragraph
        - used to add a pragraph
        - a block tag
        - `<p>this is para </p>`
    - div
        - divsion used to group the contents
        - can be also used for grouping the tags
        - adds new line character after finishing the data
        - a block tag
        - `<div>this is div 1</div>`
    - span
        - used to add textual information
        - does not add new line character after finishing the data
        - an inline tag
        - `<span>this is a span</span>`
- **formatting tags**
    - bold
        - used to make the content bold
        - ```
          India is my country.
          India is my country.
          ```
    - italic
        - used to make the content italic
        - `<div><i>India</i> is my country.</div>`
    - underline
        - used to make the cotent underline
        - `<div><u>India</u> is my country.</div>`
    - strike
        - used to make the content strike through
          ```

        - ```
          <strike>China</strike> is spreading Covid-19.
          ```
    - monospaced
        - used to create a monospaced content
        - ```
          <div><tt>China</tt> is spreading Covid-19.</div>
          ```
    - superscript
        - used to add the content above the line
        - ```
          x<sup>2</sup>
          ```
    - subscript
        - used to add the content below the line
        - ```
          <div>x<sub>2</sub><div>
          ```
    - pre
        - used to add pre-formatted text
        - e.g.

          ```
          <pre>
          ```

          this is already formatted text

          </pre>
    - marquee
        - used to show the moving text
        - e.g.
- **lists**
    - ordered
        - used to create an ordered list
        - types
            - 1: latin numbers
            - A: upper case letter
            - a: lower case letter
            - I: upper case roman numbers
            - i: lower case roman numbers
        - e.g.

          ```
          <ol type="1">
          <li>this is list item 1</li>
          <li>this is list item 2</li>
          <li>this is list item 3</li>
          </ol>
          ```

- unordered
  - used to create unordered list
  - types
    - disc: filled circle
    - circle: empty circle
    - square: square
  - e.g.

```
<ul type="disc">
<li>this is list item 1</li>
<li>this is list item 2</li>
<li>this is list item 3</li>
</ul>
```

- definition
  - used to add the definitions along with the information about them
  - dt: definition term
  - dd: definition definition
  - e.g.

```
<dl>
<dt>HTML</dt>
<dd>Hyper Text Markup Language<dd>
</dl>
```

- **resources**
  - images
    - used to render images in the web page
    - e.g.

```
<img src="cat.1.jpg">
```

  - audio
    - used to play audio
    - e.g.

```
<audio src="audio.mp3">
```

  - video
    - used to play video files
    - e.g.

```
<video src="movie.mov">
```

- **table**
- **anchor**
- **white space**

- br
    - used to add a line break
    - e.g.

        ```
        <br>
        <br/>
        ```

- hr
    - used to add line break with a horizontal rular
    - e.g.

        ```
        <hr>
        <hr/>
        ```

-
    - used to add a space
    - e.g.

        ```
         
        ```

- **form tags**
    - used for getting input values from user
    - tags
        - input
            - used to get single line input from user
            - types
                - **text**: get textual (characters + numbers) input from user
                - **date**: get date input from user (browser may show a calendar control to select the date)
                - **time**: get time input from user
                - **number**: get only number input from user
                - **submit**
                    - converts the input tag to a button
                    - used to send the information filled by user to the server
                - **reset**:
                    - converts the input tag to a button
                    - resets every tag inside the form
                - **button**
                    - converts the input tag to a button
                    - used to perform an action locally (form will not be submitted or reset)
                - **radio**
                    - used to get one of the options from user
                - **checkbox**
                    - used to get mulitple options from user
                - **file**:

        - used to select a file
    - **password**:
        - used to accept password from user
        - masks the characters (displays * instead of real characters)
    - **tel**:
        - used to get telephone number input from user
    - **email**:
        - used to get email address from user
- attributes
    - **readonly**: makes the input readonly
    - **required**: user must enter the value in the input tag
    - **placeholder**: shows a hint to the user
    - **maxlength**: used to limit the number of characters entered by user
    - **value**: used to show the tag's title
    - **checked**:
        - use to select an option by default
        - only used along with radio and checkbox
    - **accept**:
        - used to allow certain type of files
        - used only with type = file
- textarea
    - used to get multi-line input from user
- select
    - used to get one or more options from the given list of options

CSS

- Cascading Style Sheet
- used to decorate the web page
    - manage the shapes
    - manage the sizes
    - manage the colors
    - manage the animation
    - manage the mobile friendlyness
- not used for
    - adding programming logic in the website
    - designing web pages

Ways to add CSS in html document

- **browser default css**
    - by default, css which is provided by every browser
    - the browser default css will be browser specific
    - which is responsible for displaying the default tags
        - h1 will be rendered using biggest font size
        - ul and li will be rendered one after another in vertical orientation
    - generally, it is **not adivsible** to modify the default CSS
- **inline css**
    - adding the css rules inside the target tag using style attribute
    - not encouraged to use the inline style
    - limitation
        - needs to be repeated with every tag that requires modification
        - very difficult to manage / update the code
    - e.g.
        - <h1 style="color: red;">this is header1</h1>
- **internal css**
    - which is added internally to the page
    - must be added using style tag in head section
    - e.g.

        ```
        <style> div { color: red; } </style>
        ```
- **external css**
    - which is added outside the page
    - linked with the page using
    - e.g.

        ```
        <link rel="stylesheet" href="styles.css">
        ```

termonologies

- **css property**
  - used to modify the visual properties of a tag
  - e.g.
    - color, font-family, font-size, border etc.
- **css value**
  - value of the property to be modified
  - e.g.
    - red is a value
    - 20px is value
- **css declaration**
  - used to modify the visual appearance of the tag/UI
  - pair of css property and its value
  - property and its value get separated by colon (:)
  - multiple declarations are separated by semi-colon (;)
  - only one declarration need not require to be terminated with ; (semi-colon is optional)
  - e.g.

    > color: red; font-size: 20px;
- **css declaration block**
  - collection of multiple declarations
  - starts with { and ends with }
  - e.g.

    > { color: red; font-size: 20px; }
- **css selector**
  - used to select the target elements (tags)
  - e.g.

    > **div** { color: red; } /_ all divisions will be decorated with red color _/
- **css rule**
  - also known as css ruleset
  - pair of css selector and css declaration block
  - e.g.

    > div { color: red; font-size: 20px; }

css units

- px

    - stands for pixels
    - pixel: picture element
- percentage (%)
- em/rem
- degree

CSS Selectors types

- **type selector**
    - used to select similar type of elements
    - also known as element selector
    - e.g.

        > div { color: red; } /_ div selector will select only div tags _/

- **multiple type selector**
    - also known as a combinator selector
    - uses punctuation symbol comma (,)
    - used to select multiple types of elements
    - e.g.

        > div, p, span { font-size: 20px } /_ all divisions, paragraphs and spans will be decorated with font size set to 20px _/

- **id selector**
    - used to target an element based on the id attribute value
    - uses punctuation symbol hash (#)
    - e.g.

        > #div-3 { color: red; } /_ any element having an id div-3 will get decorated with red color _/

        div#div-3 { color: red; } /_ only div element having an id div-3 will get decorated with red color _/

- **class selector**
    - used to target element(s) based on the class attribute
    - uses punctuation symbol dot (.)
    - e.g.

        > .div-3 { color: red; } /_ any element having a class div-3 will get decorated with red color _/

        div.div-3 { color: red; } /_ only div element having class div-3 will get decorated with red color _/

- **universal selector**
  - used to apply rules on every possible element in the page
  - uses punctuation symbol star (*)
  - e.g.
    > { font-family: arial } /_ all elements will use the font as arial _/
- **attribute selector**
  - used to select element(s) based on the value of an attribute
  - uses punctuation symbol square bracket []
  - e.g.

    > input[type="submit"] { background-color: green; } /_ only input having type = submit will get green backgound _/
- **descendent selector**
  - used to select the element(s) based on the parent-child relationship
  - selects all the element(s) which are descedent [appear at any level: child, grand-child ..] of parent
  - e.g.

    > <style>

    ```
    div p { color: green }

    /* all paras will turn to green [a all paras are descendent of
    div] */
    ```

    </style>

     para 1 inside div para 2 inside div <ul> para 1 inside li   para 2 inside li
- **child selector**
  - used to select the element(s) based on the parent-child relationship
  - selects all the element(s) which are direct child element(s) of parent
  - e.g.

    > <style>

    ```
    div > p { color: green }
    ```

```
/* paras which are direct child elements of div will turn to
green*/
```

</style>

 para 1 inside div para 2 inside div <ul> para 1 inside li   para 2 inside li

- **general sibling selector**
  - used to select element(s) based on the levels they appear on
  - uses punctuation symbol tild (~)
  - e.g.

    > p ~ span { color: red; } /_ select spans appearing on the same level as that of para and after paragraph _/

- **adjacent sibling selector**
  - used to select element(s) based on the levels they appear on
  - uses punctuation symbol plus (+)
  - e.g.

    > p + span { color: red; } /_ select spans appearing on the same level immediately after paragraph _/

- **pseudo selector**
  - **pseudo class**
    - keyword added to a selector that specifies a special state of the selected element(s)
    - e.g.

      > div:hover { color: green; } /_ div will be decorated with green color only when mouse goes on top of it _/

  - **pseudo element**
    - is a keyword added to a selector that lets you style a specific part of the selected element(s)
    - e.g.

      > p::first-letter {

      ```
      color: red;
      ```

      }

      /_ only the first character will be decorated with red color _/

CSS Box Model

- every element in CSS is rendered as a box with following properties
  - border
    - the bounding box for the element
    - has following properties
      - style
      - width
      - color
      - radius
  - padding
    - gap between the border and content
  - margin
    - gap outside the border

CSS Positions

- **static**
  - default position
  - decided by the code structure
  - top, left, bottom and right properties will be ignored
  - e.g.

```
button {
  position: static;
}
```

- **relative**
  - the new position (by setting top, bottom, left and right) with respect to the default position
  - e.g.

```
button {
  position: static;
  top: 10px;
  left: 10px;
}
```

- **animation**
  - transform
  - transition

- way to create an object
- JSON supports
    - object
        - collection of key-value pairs
        - e.g.

```
const person = {
  name: 'person1',
  email: 'person1@test.com',
  age: 40,
}
```

    - array
        - collection of objects
        - e.g.

```
const persons = [
  { name: 'person1', email: 'person1@test.com' },
  { name: 'person2', email: 'person2@test.com' },
  { name: 'person3', email: 'person3@test.com' },
  { name: 'person4', email: 'person4@test.com' },
]
```

creating object using Object

- Object is a root function provided by JS
- everything in JS is an Object

node

module

- any javascript file having an extension .js
- the NodeJs embeds an object named module in every module to present the current module
- the module object has following properties
    - id: the identification of the module
    - path: the file which is representing this module
    - exports:
        - contains the list of functions/variables/constant which can be exported from the current module

Computer Language

- definitions
    - set of instructions (algorithm)
    - implementation of algorithm
    - helps us to interact with hardware
    - medium of communication with hardware
- types
    - based on the level
        - **low level**
            - binary (0s and 1s)
        - **middle level**
            - interacts with CPU
            - Assembly language
            - opcodes: operation code => binary
            - e.g. ADD A, B
        - **high level**
            - developer can write human understable code
            - compiler or interprter convers the human understandable to machine (CPU) understandable (ASM)
            - e.g. C++, Java, Python
    - based on how the application gets generated
        - **compiled language**
            - compile: converting human understable to machine (CPU) understandale
            - compiler: program which does compilation
            - executable:
                - program which contains only ASM instructions (machine understandable)
                - native applications
                - always platform (OS) dependent
                - faster than interpreted program
            - requires compiler
            - the entire program gets converted into executable
            - if program contains error, compiler detects these error at compilation time
            - e.g. C, C++
        - **interperted language**
            - interpretation: which converts the human understandable to machine (CPU) understandable line by line
            - interpreter: program which does interpretation
            - no executable gets generated
            - if there is any error, it will get detected at the run time

        - program will be always platform (OS) independent
        - programs will be always slower than native applications
        - e.g. html/CSS, JS, bash scripting
    - **mixed language**
        - shows behavior from both (compiled as well as interptreted)
        - uses compiler as well as interpreter
        - e.g. Java, **Python**

JavaScript

- is a scripting language
- is an object oriented programming language
- is functional programming language
- is used for adding dynamic behavior (e.g. clicking a button) in the website
- is loosely typed language
    - there is not type checking
    - the data types are inferred
    - the data types are dynamically assigned
- does NOT support pointers

JS Fundamentals

- **rules**
    - the semicolon is optional when one statement is written on one line

```
console.log('this is JS tag in head section')
console.error('this is an error')
```

    - semicolon is required when multiple statements are written on one line

```
console.log('this is JS tag in head section')
console.error('this is an error')
```

    - conventions
        - use camel case when declaring function, variables or classes
        - i.e. start the name with lower case and use upper case for first letter of meaningful word
        - e.g.
            - firstName
            - printPersonInfo()
- **variable**
    - is a placeholder to store a value in memory

- it is a mutable
- to declare a variable use **let** keyword
- the variable must be declared without a data type
- e.g.

```
let firstName = 'steve'
let salary = 10.6

let age = 40

// can update the value
age = 41
```

- **constant**
  - is a placeholder whose value **CAN NOT** be changed
  - this is immutable (readonly)
  - to declare a constant use **const** keyword
  - the constant must be declared without a data type
  - e.g.

```
const pi = 3.14

// can not update the value of a constant
// pi = 100
```

> ALWAYS TRY TO PREFER CONSTANT THAN A VARIABLE

- **pre-defined objects**
  - console
    - object that represents the browser console
    - methods
      - log() : used to print debugging messages on the browser's console
      - info() : used to print information on the browser's console
      - warn() : used to print warning messages on the browser's console
      - error() : used to print error on the browser's console
  - window
    - object that represents the browser's window (UI)
    - this is a default object used when calling a method
    - methods
      - alert() :

        - prompt() :
        - confirm() :

- **Pop ups**
    - alert
        - used to show a message to the user on web browser's window
        - e.g.

```
// window.alert('this is an alert')
alert('this is an alert')
```

    - prompt
        - used to take an input from user
        - e.g.

```
const username = window.prompt('Enter your name')
console.log('user name = ' + username)
```

    - confirm
        - used to get input in terms of boolean answer to a question
        - e.g.

```
const answer = window.confirm('Do you want to have break?')
if (answer) {
  console.log('lets take a break of 10 minutes')
} else {
  console.log('lets continue')
}
```

- **pre-defined values**
    - **undefined**
    - **NaN**
        - Not a Number
        - NaN has data type as number
        - does not represent a valid number
        - e.g.

```
// NaN
console.log(10 * 'test1')
```

- **Infinity**
  - has a data type as number
  - e.g.

```
// Infinity
console.log(`10 / 0 = ${10 / 0}`)
```

- **data types**
  - all data types in javascript will be inferred
  - the data type will be decided by JavaScript by inspecting the current value in the variable
  - types
    - **number**
      - represents whole numbers and floating point (decimal) numbers
      - e.g.

```
// number
let num = 100
console.log('data type of num = ' + typeof num) // number

// number
let salary = 10.5
console.log('data type of salary = ' + typeof salary) //
number
```

    - **string**
      - collection of characters
      - string can be declared using
        - single quotes (')
        - double quotes (")
        - back quotes (`)
      - e.g.

```
// string
let firstName = 'steve'
console.log('data type of firstName = ' + typeof
firstName) // string

// string
let lastName = 'Jobs'
console.log('data type of lastName = ' + typeof
lastName) // string

// string
let address = `
      address line 1,
      address line 2,
  `
console.log('data type of address = ' + typeof address)
// string
```

- **boolean**
  - represents only true or false values
  - e.g.

```
// boolean
let canVote = false
console.log('data type of canVote = ' + typeof canVote)
// boolean
```

- **undefined**
  - in JS, undefined is both: data type as well as pre-defined value
  - prepresents a variable without having initial value
  - e.g.

```
// undefined
let myvar
console.log('myvar = ' + myvar) // undefined
console.log('data type of myvar = ' + typeof myvar) //
undefined
```

- **object**

- **statements**
  - the smallest unit that executes
  - types

        - assignment
        - declaration
        - comment

- **operators**
    - mathematical operators
        - addition (+)
            - the plus operator works as
                - mathematical addition when bothe params are numbers

```
// 30
console.log(10 + 20)
```

                - string concatination opetator when one of the operands is a string

```
// test1test2
console.log('test1' + 'test2')
```

                - when one of the operands is a string and other is not a string, then all the params get converted to string data type

```
// 1020
console.log(10 + '20')
```

        - division (/)
            - mathematical division
            - e.g.

```

```

        - multiplication (*)
            - mathematical multiplication of two operands
            - the answer will be always a number
            - e.g.

```

```

```
// 200
console.log(10 * 20)

// NaN
console.log('test1' * 'test2')

// 400
console.log('10' * 40)

// 400
console.log('10' * '40')
```

- modulo (%)
- subtraction (-) :

- comparison operators
  - double equals to (==)
    - also known as value equality operator
    - checks ONLY the values of operands
    - e.g.

```
// true
console.log(50 == 50)

// true
// '50' is having a value of 50
console.log(50 == '50')
```

  - triple equals to (===)
    - also known as identity equality operator
    - checks both the value as well as the data types of the operands
    - always prefer === over ==
    - e.g.

```
// true
console.log(50 === 50)

// false
// 50 is a number while '50' is a string
console.log(50 === '50')
```

        - not equals to (!=) :
        - not equals to (!==) :
        - less than (<) :
        - greater than (>) :
        - less than equals to (<=) :
        - greater than equals to (>=) :
    - logical operators
        - and (&&) :
        - or (||) :

- **type conversion**
    - anything to string
        - any data type can be converted into string by concatinating with plus operator
        - e.g.

```
// '10'
console.log(10 + '')
```

    - string to number
        - parseInt
            - convert string to integer number (by discarding the decimal precision)
            - e.g.

```
// 10
console.log(parseInt('10'))

// 10
console.log(parseInt('10.60'))

// 10
console.log(parseInt('10test'))

// NaN
console.log(parseInt('test10'))
```

        - parseFloat

        - convert a string to a decimal number (by keeping the decimal precision)
        - e.g.

```
// 10
console.log(parseFloat('10'))

// 10.60
console.log(parseFloat('10.60'))

// 10
console.log(parseFloat('10test'))

// NaN
console.log(parseFloat('test10'))
```

- **function**
    - a block of code which can be reused
    - reusable block of code having a name
    - types
        - empty function
            - function with no code in the body
            - e.g.

```
// empty function
function function0() {}
```

        - parameterless function
            - which does not accept any parameter
            - e.g.

```
// parameterless function declaration
function function1() {
  console.log('inside function1')
}

// function call
function1()
```

        - parameterized function

        - which accepts at least one parameter
        - e.g.

```
function function2(param) {
  // param = true
  console.log(`inside the function2`)
  console.log(`param = ${param}, type of param =
${typeof param}`)
}

function2(20)
function2('test1')
function2(true)
```

    - **variable length argument function**
        - a function which can accept variable length of arguments
        - every function in JS receives a hidden parameter named arguments
            - which contains a list of all the arguments passed while the function call
        - e.g.

```
function add() {
  // console.log('inside add')
  console.log(arguments)

  let sum = 0
  for (let index = 0; index < arguments.length;
index++) {
    sum += arguments[index]
  }
  console.log(`addition = ${sum}`)
}

add(10, 20)
add(10, 20, 30)
add(10, 20, 30, 40)
```

- parameters
    - the parameter(s) of a function can not have static data type(s)
    - the number and type of parameters will be controlled by the caller instead of the function
    - caller can pass

    - same number of parameters
    - less number of parameters than expected
        - the missing arguments will be treated as undefined
    - more number of parameters than expected
        -
        - the extra parameters will be discarded

- every function in JS receives two hidden parameters
    - **arguments**: used to get all the arguments in an array
    - **this**: used to point the current object
        - **this** points to window in a function inside javascript in html
        - **this** porint to Object if called oustide the html

- e.g.

```
function function1(n1, n2) {
  console.log('inside function1')
}

// n1 = 10, n2 = 20
function1(10, 20, 30)

// n1 = 10, n2 = 20
function1(10, 20)

// n1 = 10, n2 = undefined
function1(10)

// n1 = undefined, n2 = undefined
function1()
```

- default parameters
    - also known as optonal parameters as you may not pass the value to these parameters
    - one or more paramters may be declared with a default value
    - the default value will be used if the caller has not passed the argument for the optional paramters
    - e.g.

```
function multiply(num1, num2 = 10) {
  const multiplication = num1 * num2
  console.log(`multiplication = ${multiplication}`)
}

// num1 = 40, num2 = 100
multiply(40, 100)
```

```
// num1 = 40, num2 = 10
multiply(40)
```

- function alias
  - giving a function another name
  - same function can be called with original function or the function alias
  - e.g.

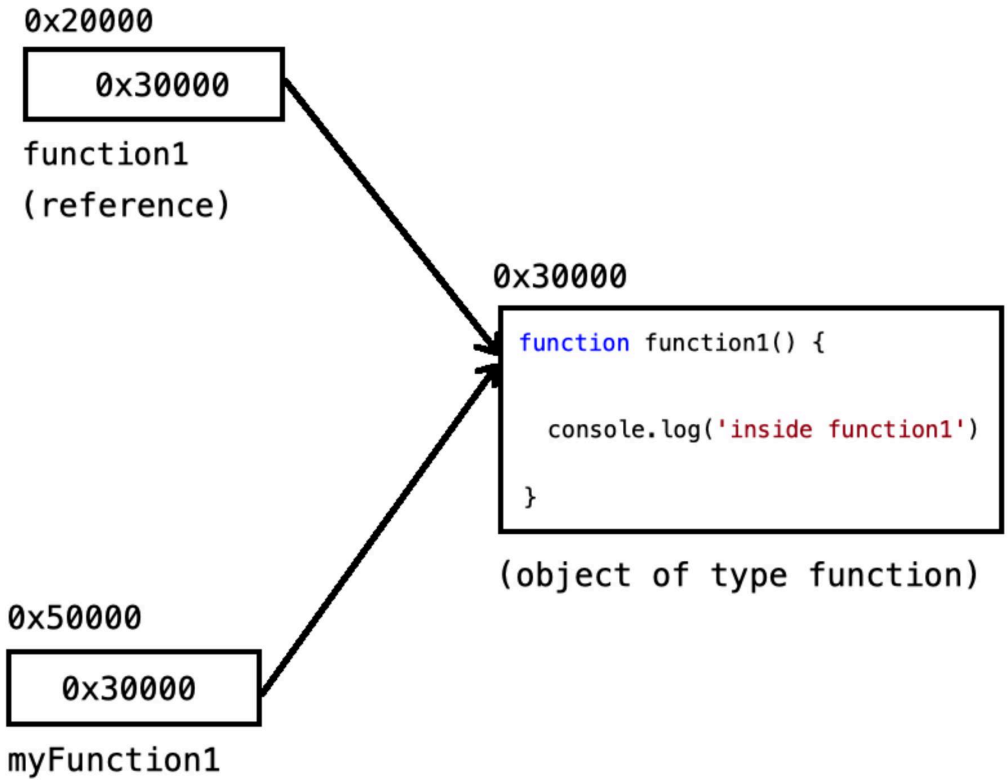
```
function function1() {
  console.log('inside function1')
}

// calling the function1 by its orginal name
function1()

// function alias
const myFunction1 = function1

// calling the function1 by its function alias
myFunction1()
```

```
function function1() {
    console.log('inside function1')
}
```




```
const myFunction1 = function1
```

- **collection**
  - collection of values (similar to array of values in C and C++)
  - properties
    - **length**
  - methods
    - **push**
      - used to append a value at the end of the collection
      - e.g.

```
const numbers = [10, 20, 30]

// [10, 20, 30, 40]
numbers.push(40)
```

- **html + JS**
- **predefined functions**
    - typeof() : used to get the data type of a variable

JS as functional programming language

- in JS, function is considered as first class citizen
    - function can be called by passing another function as an argument
    - a variable can be created for a function (function alias)
        - a function can be considered as a variable
    - one function can be considered as a return value of another function

OOP JS

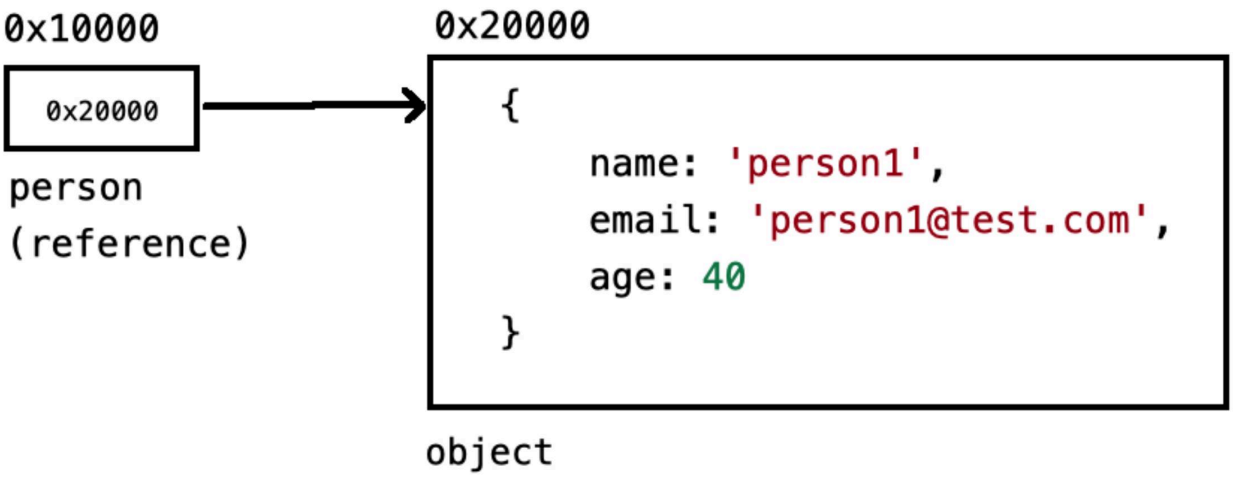
- JS is object oriented programming language
- object can be created by
    - using JSON
    - using Object
    - using constructor functions
    - using keyword class

object

- similar to instance of a class in other language
- collection of key-value pairs
- keys are also known as properties
- e.g.

```
const person = {
  name: 'person1',
  email: 'person1@test.com',
  age: 40,
}

// properties - name, email, age
// values     - person1, person1@test.com , 40
```



- to access value of a property
  - use subscript ([] syntax)
    - used when a property is having special character like space
    - e.g.

```
console.log(`name: ${p1['name']}`)
console.log(`email: ${p1['email']}`)
console.log(`age: ${p1['age']}`)
```

```
const person = {
  'first name': 'steve',
  'last name': 'jobs',
}

console.log(`first name = ${person['first name']}`)
console.log(`last name = ${person['last name']}`)
```

  - use dot (.) syntax
    - e.g.

```
console.log(`name: ${p1.name}`)
console.log(`email: ${p1.email}`)
console.log(`age: ${p1.age}`)
```

    - can not be used when a property has a special character like space

JSON

- JavaScript Object Notation

React

- JS library to build user interface
- developed and donated by facebook (meta)
- facebook has made the react open source in 2013

commands

- install yarn

```
> npm install -g yarn
```

- create a new project

```
> npx create-react-app newapp
```

- run a react application

```
> cd newapp
> yarn start
```

- test a react application

```
> yarn test
```

- build the react application

```
> yarn build
```

file hierarchy

- **node_modules**
  - contains the node packages required to run the react application
- **public**:
  - contains the static data
  - e.g. css, index.html, images
- **src**:
  - contains the application source code
  - index.js:
    - entry point to the application

        - application starts with index.js
    - App.js
        - contains the first/startup component
- **.gitignore**
    - contains the list of files/directories which are not needed to add to the git repository
- **package.json**
    - contains the dependencies
- **README.md**
    - contains the read me instructions for the application
- **yarn.lock**
    - contains the packages versions used in the application
    - will get automatically created

component

- reusable entity defined by react
- created by
    - using a function
    - using a class
- may represent
    - an entire page
    - a part of a page

react routing

- documentation
    - https://reactrouter.com/docs/en/v6
- installation

```
> yarn add react-router react-router-dom
```

blogs

- create and configure the application

```
> npx create-react-app blogs
> cd blogs
> yarn add react-router react-router-dom axios moment moment-timezone
```

- packages
    - react-router and react-router-dom
        - needed for adding page/component switching functionality
    - axios
        - used for calling the REST apis

- moment and moment-timezone
    - used for time/timezone conversion