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 = 10.60

    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 = 10.50
  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
  - 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.

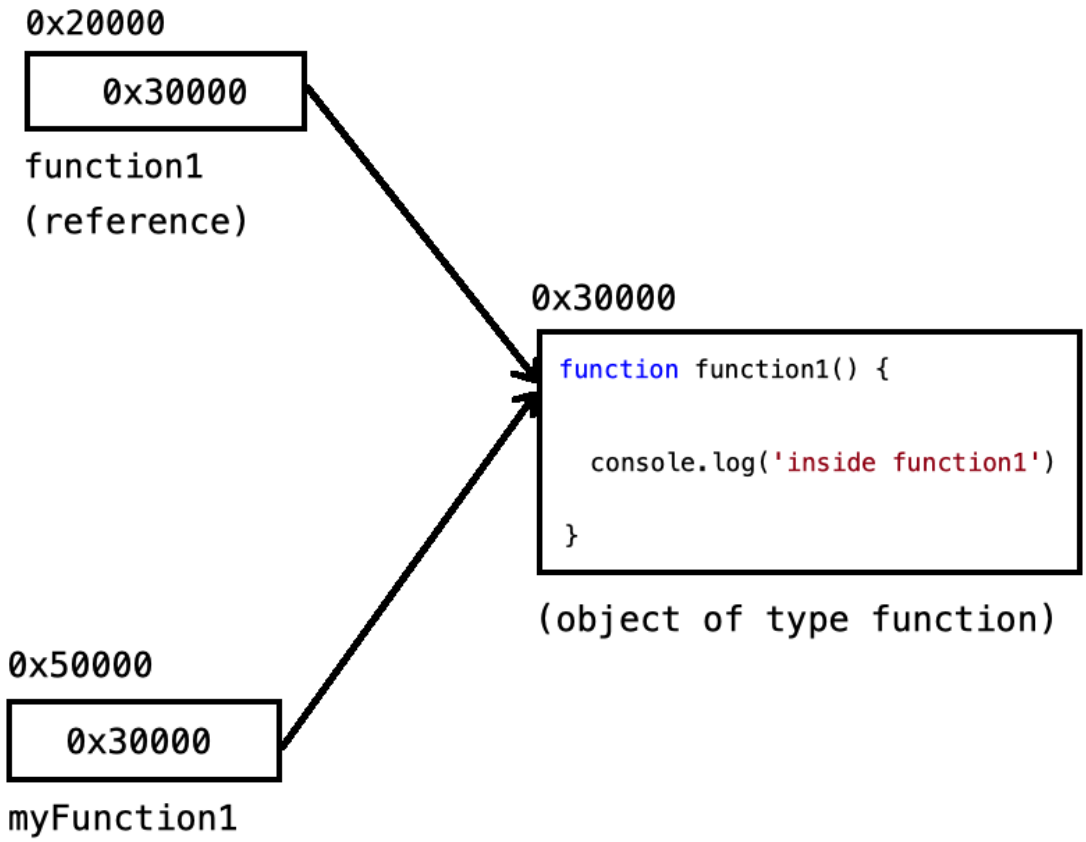
```
function function1() {
  console.log('inside function1')
}

// 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

OOP JS

JS as functional programming language