Javascript
==========
- programming language
- to provide behaviour and perform logical operations
- to execute Javascript we need a run time environment
- Node js is used to provide this run time environmnet
- Run Time environment includes, javascript engine, it compiles the code and execute it
- By default browser has JS engine
    eg: Chrome: v8
        Mozilla: spider monkey
- extension for javascript file : .js
- console.log() - is used to print output in the terminal

DataTypes
==========
1) primitive Data DataTypes
        String - text
        Number - integers and decimal values
        boolean - true/false
        undefined - variable declared, but no value assigned
        null - no value at all
2) Reference Type
        Object:
                js Object
                Array
** Javascript is a dynamically typed language: we dont have to implicitly specify the
     type of value need to store in a variable. Js automatically recognize its type
** var keyword is used to declare a variable
        variable is used to store a value

Basic commands in JS
==========
1) console.log() : to print output in terminal
2) var : to declare a variable inorder to hold a value
3) typeof :to return data type of that specific variable/value

Template literal

- used to append dynamic data with string
syntax: `${variable}`

Assignment Opeartor

= to assign values, from right to left;
eg: var num = 5

Arithmetic Operators

+, -, *, /, %
% - modulus operator, it return the reminder

== and === : comparison Operators

== only checks the value
=== checks both value and data type
> greater than
< lessthan
>= greater or equal to
<= lessthan or eqaul to
!= not eqaual to
!== not equal to with type check

Logical Operators
==========
&& - if all conditions are true (AND)
|| - if any condition is true (OR)
! - if condition not satisfy

Increment and decrement operator
---
 increment or decrement value by 1
++ Increment
-- decrement

String Concatenation

+ is used for string concatenation (string combining)
if any value on left or right of the + symbol is string, then javascript convert the other value into string also

conditional statements/ decision making satetements
---
1) if
2) if else
3) else if ladder
4) while
5) switch

if
--
```
if(condition){
    // if condition true, it execute this code
}
```

if else
=======
```
if(condition){
```

```
    // if condition true, it execute this code
}
else{
    // if condition fails, it execute this code
}

else if ladder
-----------------
- used to check multiple condition
- if one condition fails, it check second condition, if second condition fails, it
check third condition, so on
 syntax:

 if(condition1){
    // block of code to be executed
 }
 else if(condition2){
    // block of code to be executed
 }
 else if(condition3){
    // block of code to be executed
 }
 else{
    // code of block to be executed if no codition is true
 }

 while
 ------------
 - Repeatedly execute a block of code as long as the specified condition is true
 syntax:

 initialize variable
 while(condition){
    // block of code to be executed
 }
 break: can be used to exist from the loop

 do while
 -----------------
 - it execute the block of code atleast once and then repeatedly execute the block
of code as long as
    specified condition is true
    syntax:

    do{
        // code to be executed
    }while(condition)

Ternary Operators
-------------------
```

simplified form of if else/ else if ladder

syntax:
condition?"code to execute":"code to execute"
example:
var num=46;
num<50? "Num is less than 50" : "Num is greter than or equal to 50"

shorthand operator

+= examaple x+=i x=x+i
-= example x-=i x= x-i

NaN = Not A Number
parseInt(num) : used to remove decimal point from a number
to find the length of a string = string.length
to find the lenngth of a number = number.toString().length

for loop

- to repeatedly execute a block of codes long as the
 specified condition is true
 syntax:
 for(initializing a variable; condition; inccrement/decrement the variable){
    // block of code tobe executed
 }

 nested loop

  - loop inside another loop

syntax:
 for(initializing a variable; condition; inccrement/decrement the variable){
     for(initializing a variable; condition; inccrement/decrement the variable){
    // block of code tobe executed
    }
 }

 switch

- The switch statement is used to perform different actions based on different
conditions
- The switch statement is used to select one of many code blocks to be executed

syntax:
switch(expression){
    case x:
        // code block to execute;
        break;

```
    case y:
        // code block to execute
        break
    case z:
        // code block to execute
        break;
    default:
        // code block to execute
}
```

Functions
==========
Functions are reusable blocks of code designed to perform a particular task.

Functions has two parts:
1) function definition
2) function call

syntax for function definition

```
function function_name(argument1, argument2, ..){
    // block of code to be executed
}
// arguments are not mandatory
```

function call

```
function_name(argument1, argument2, ..)
```

Function Declarartion
---
above one

Function expression

```
var function_name = function(arguments){
    // code of block to be executed
}
function_name(arguments)
```

return
---
return is used to take result of a function, outside the function block

Camel case: it is a naming convention in which each word within a compound word is
            capitalized, except for the first word.
            this style is commonly used in programming for naming variables,
            functions, methods, objects, etc
            addTwoNumbers

Pre-defined functions in js

- Already defined functions, we only need to call that function. No need to defined
- it is globally accessible
    1) console.log()
    2) alert()
    3) typeof()
    4) parseInt()

Self-invoking functions

There is no need to call this function to execute, It execute by itself
syntax:

```
(function(){
   // code of block to be executed
})();
```

setInterval()

- this function continously execute in particular interval of time
syntax:

```
setinterval(function(){
    // code block to execute
}, time in milliseconds)
```

setTimeout()

- this function execute only once, when the timeout reaches

```
setTimeout(function(){
    // code to be executed
}, time in milliseconds)
```

callback functions

A callback function is js is function , that is passed as an argumnet to another function
and it is executed after a certain task is finished.
Callbacks are a way to ensure that certain code doesn't execute until
a specifc task is completed, allowing for better control over the flow of our program

Nested function

A function is defined within another function

Arrow Function

- also called fat arrow function
- it is a concise way to write function expression
- It provide compact syntax compared to traditional function expression

```
syntax:
variable = (arg1, arg2,..)=>{
    // code of block to be extecuted
}
eg:
var sum = (a,b)=>{
    console.log(a+b)
}
 sum(2,3)
```

- it provide shorter syntax for single expression
- ie: if the arrow function body consists of a single expression, we can omit the curly braces and the return keyword. The result of expression will be implicitly/automatically returned

```
    eg:
    var multiply=(a,b)=>a*b;
    multiply(3,5)
```

Hoisting

Hoisting is a Javascript mechanism, where variables and function declarations are moved to
the top of their containing scope before the code executes.
This means that we can access variables and functions before they are declared;

While Javascript execute a program, mainly it has 2 phases

1) memory creation phase
    - this the first step in executing a js program
    - in this phase, javascript allocate memory for variables and functions
    - assigning values and function invocations are not happening in this phase
2) execution phase
    - after memory creation phase, execution phase starts
    - assigning values and function calling/invocation happens in this phase

Functions expressions and arrow functions cannot be hoisted
because, in arrow function and function expression, the functions are assigned to a variable.
so in memory creation phase, this variables are added to the memory with value undefined,
only at the execution phase, the corresponding function definitions are assigned to that varaibles

callback hell

- it is also called pyramid of doom
- it is term in javascript programming to describe a situation where callback functions are nested within multiple level of operations

- simply nested callback
- if any one of the callback function fails, the entire code fails

```
function function1(function(result1)){
  function2(result1, function(result2){
      function3(result2, function(result3)){
          ----------
      }
  })
}
```

mechanis, to solve callback hell
1) using promise
2) using async await

Arrays

- special type of variable that can hold more than one value at a time
- syntax: var varaible_name=[value1, value2, value3,...]
- values are seperated by comma
- eg: var fruits=['apple','orange','kiwi']
- eg: var marks=[23,45,34]
- array can hold multiple data type values
- eg: var arr=["innova",56, true]
- each element in an array is identified by index value
- index is start from 0
- .length can be used to find length of an array
- eg: fruits.length
- index range from 0 to length-1

main 4 built in method for array operations:
1) push: insert an item into last position of an array
2) pop: remove an item from the last position of an array
3) shift: remove an item from the first position
4) unshift: insert an item into first position

Different ways to access elements in an array
---
1) for loop

```
for(var i=0; i<arr.length; i++){
   console.log(arr[i])
}
```

2) looping through index

```
    for(var i in arr){
        console.log(arr[i])
    }
```

3) looping through each item

```
    for(var item of arr){
        console.log(item)
    }
```

Different array methods

---

1) length
2) indexOf : to return the index of particular item
    if item present in the array, it return its index value
    if item is not present, it return -1

var, let, const

==========

The above threes are used for variable declarartion. But they have different behavior in terms
of scope, hoisting and mutability

scope: boundary in which variables can be accessed

var

=======

1) scope: Function scoped variables are declared with 'var' are scoped to the function
in which they are declared, or globally if declared outside of a function
(function scoped or Global scoped)
2) Hoisting: 'var' varaibles are hoisted to the top of their scope and js initialize its
value as undefined
3) Re Declaration: we can re-decalre var inside same scope. also we can re-assign values

let

======

1) scope: Block scoped, variable declared with let are limited to block in which they are
declared (eg: {})
2) Hoisting: Let are hoisted to the top of their block, But are not initalized with undefined value. JS kept this let variables in temperal dead zone. So accessing them
before delcaration results in 'reference error'
3) Re-Declarartion: Not allowed within the same scope, but value can be re-assigned

const

==========

1) Scope: same as let. variable declared with const are limited to block in
 which they are declared (eg: {})
2) Hoisting: Let are hoisted to the top of their block, But are not initalized with undefined value. JS kept this let variables in temperal dead zone. So accessing them
before delcaration results in 'reference error'
3) Re-delcaration : Not allowed within the same scope

4) Assignment: Must be initialized at the time of declaration and cannot be re-assigned

sort

- in built js method used for sorting an array
syntax:
for single digit array to sort in ascending order:
arr.sort()

for more than one digit array in ascending order:
arr.sort((a,b)=>a-b)

for more than two digit array in descending order
arr.sort((a,b)=>b-a);

for sorting string array in alpahbetic order:
arr.sort();

for sorting string array in descending order:
arr.sort((a, b) => b.localeCompare(a))

forEach()

- used to iterate over each element in an array

```
arr.forEach((element)=>{
    console.log(element)
})
```

difference between array iteration using for loop and forEach
---
- Both are used to iterate over each element in an array
- in for loop we can apply 'break' to exit from the loop, if specific condition reaches
- we can have better control over the elements in array (i value can be define on our needs)
- we can manually set the the value of i and length

map()
---
map method is used to create a new array by applying a function to each element in an existing array. It doesn't modify the original array, but instead it returns a new array
containing the results of applying function to each element.

```
const result = arr.map((elemen)=>{
 // code block to be executed
})
```
eg: let arr = [1,2,3,4,5];

```
const result = arr.map((ele)=>ele*2)
// result array contains the new array wich hold
the results of the function

filter()
------------
filter method is used to create a new arry containing only
the elements that satisfies a specific condition
- it does not modify the original array, instead it create
a new array
syntax:
const result = arr.filter((element)=>{
    // filter condition
})

reduce()
------------
reduce method execute function over all elements in the
array and accumulates them into a single value
syntax:
const result = arr.reduce((accumulator, currentvalue)=>{
    // code to be executed
},initialvalue of accumulator)

eg:
sum of numbers in an array

find()
-----------
find method is used to retrive the first element in an
array that satisfies the provided condition
- it iterate over each element in the array and returns the
first element for which the condition is true
- if no element in the array satisfies the provided condition
'undefined' is returned for

includes()
------------
it is a built in method in javascript for arrays.
 It is used to check whether an array contains a specific element.
It returns trus, if the element is found and false, if the element not found
syntax:
arr.inculdes(value{to check})
returns true or false

some()
----------
some() method in js is used to check whether atleast one element in an array
 satisfies specific condition
 - it return true or false
```

```
syntax: arr.some((element)=>{
        // condition to check
})

slice
----------
- slice method is used to extract a section of an existing array and it
returns a new array
- syntax:
    arr.slice(start, end)
    end: it will not take the end index, just one value behind the end index,
    end index in not mandatory
    if no end index , it take all values upto end

splice
---------
splice method is used to add or remove array elelements
it modifies the original Array
syntax:
array.splice(index, count, item1, item2,...)

reverse()
----------
reverse array elements
syntax: arr.reverse()

Nested array
---------------
array inside another array
eg: let arr = [1,2,[3,4],5,6,[7,8]];
        arr1= [1,2,1,2,1,2,3]]

flat()
---------
flat method is used to convert nested array into specific depth

let x= [1,[1,[1,[1,[1]]]]]
x.flat(Infiity) - if depth is unknown we can give depth as Infinity

Linear Search
--------------------
- also known as sequential Search
- Simple algorithm it uses, that checks each element in an array sequentially
 until the desired element is found or the array ends
 - linear search can be applied on un-sorted array

 Binary Search
 -----------------
 - Binary search is an efficient algorithm for finding an item from a sorted array
```

```
 - It works repeatedly by dividing the array until, until specific item found
 - first we find the mid of the array,
 - if the value of target need to find is less than the mid, search continues in
the first half
 - if the value of target is greater than the mid, search continues in the second
half
 - if the target value is mid, then it returns the value

 closure property
 --------------------
 Consider we have an outer function and inside that outer function,
 there is an inner function.
 The inner function has access to its own variables, its outer function
 variables and globally decalred variables

 String methods
 --------------------
 1) toLowerCase(): to convert string to lowercase
 2) toUpperCase(): to convert string to uppercase
 3) trim(): to remove white space from start and end of a string
    trimStart(): to remove white space from start
    trimEnd(): to remove white space from end
4) startsWith: to check whether a string starts with
    specific character/word. It is case sensitive
5) endsWith: to check whether a string ends with
    specific character/word. It is case sensitive
6) includes : to check whether a string contains
    specific character/word. It is case sensitive
7)charAt(index): to return charcter at specifc index;
8) concat: for string concatenation/joining
        str1.concat(str2)
9) substring(startIndex, endIndex)
10) split: it returns an array
        split(seperator)
 ** join( ): to convert array into string
       join(sperator)

javascript objects
----------------
- javascript objects are collection of key value pairs
- it provide more clarity to the data stored
- syntax: var variable_name = {
    // key:value pairs
}
 var arr = ['john',28,true]

 var person = {
    name:'john',
    age:28,
    isMajor:true
```

```
 }

value can be string, integer, boolean, object, array, function
comma is used to seperate each key-value collection

Different ways to create an Object
--------------------------
1) Using object literal
    var car={
        name:"innova",
        manufacturer:'Toyota',
        price:2400000
    }

2) using new Object() syntax
    let person = new Object();
    person.name= "Todd";
    person.age= 27;
    person.email='abc@gmail.com'

Different ways of accessing object properties
---------------------------------------------
1) Dot notation
    eg: console.log(person.name)
2) Bracket Notation
    eg: console.log(person['name']);

Adding new properties/keys
--------------------
1) Dot
    person.mobileNumber = 9744468168

2) Bracket
    person['mobileNumber'] = 9744468168

3) Object.assign(target,{key:value})

Deleting a property/key
----------------------
delete object_name.key
eg: delete person.age

Example of Object with methods
------------------------------
let calculator = {
    sum:function(a,b){
        return a+b
    },
    multiply:function(a,b){
        return a*b
```

```
    }
}

Checking for a particular property
------------------------------
'key_name' in object_name
// return true or false

For getting all keys
for(let key in object_name){
    console.log(key)
}

Update the value of a key
-------------------------
object_name.key = "newValue"
object_name['key']= "newvalue"

Nested Object
------------
object inside an object
eg:
let person = {
    firstName:"John",
    lastName:"Doe",
    address:{
        street:'123 main st',
        city:'Newyork',
        zipcode:34523
    }
}
// to access city
person.address.city

Get all keys
-------------
Object.keys(object_name)
it returns an array

Get all values
-------------------
Object.values(object_name)
it returns an array

Get combination of key value pairs
----------------------------------
Object.entries(object_name)
it returns an array

Object.seal()
```

This method is used to control the mutability of object
- It prevent adding or removing properties
- It allows modification for values of existing properties

Object.freeze()

This method is used to control the mutability of object
- prevent adding, removing and modifying the properties

Array of objects

- each element in an array is an object

eg:
```
var student=[
    {name:'john', branch:'cse'},
    {name:'todd', branch:'eee'},
    {name:'stachia', branch:'ece'}
]
```