

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 environment
- 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 don't 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 in order 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 Operator

= 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
< less than
>= greater or equal to
<= less than or equal to
!= not equal 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 statements

- 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 condition 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 continuously 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 argument 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 specific 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 variables

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 variable_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 three are used for variable declaration. 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' variables are hoisted to the top of their scope and js initialize its value as undefined
- 3) Re Declaration: we can re-declare 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 initialized with undefined value. JS kept this let variables in temporal dead zone. So accessing them before declaration results in 'reference error'
- 3) Re-Declaration: 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 initialized with undefined value. JS kept this let variables in temporal dead zone. So accessing them before declaration results in 'reference error'
- 3) Re-declaration : 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 alphabetic 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 which hold
the results of the function
```

filter()

filter method is used to create a new array 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 retrieve 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 true, if the element is found and false, if the element not found

syntax:

```
arr.includes(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 is not mandatory

if no end index, it takes all values up to end

splice

splice method is used to add or remove array elements

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(Infinity)` - 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 separate 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'}  
]

eval()  
-----

eval() is a function used to evaluate or execute a string in javascript

Exception Handling  
-----

exception:Run time errors

- try-catch block is used to handle run time error
- try: Code that might commit errors
- catch: what to do if error occurs
- finally (not mandatory): control will come to this block, in both the cases

spread and rest operator (...)  
-----

spread operator:

The spread syntax expands iterables (like array or object) into individual elements where it is used.

eg:  
var array1=[1,2,3,4];  
var array2 = [...array1,5,6,7,8]  
  
var obj1={a:1,b:2}  
var obj2={...obj1,c:3,d:4}

Rest operator:

- mainly used for passing arguments to function
- we are not sure about,how many arguments the function expecting



## Array Destructuring

-----

Array destructuring in JS used to unpack values from array and store it into a variable

```
eg: var arr=[1,2,3]
let [a,b,c]=arr;
console.log(a);
console.log(b);
console.log(c)
```

## Object Destructuring

-----

Object destructuring is used to unpack properties value in to variable

- variable must be key name

## OOPS

-----

- Object oriented programming system

procedural programming: programs created using functions and variables

object oriented programming: It make use of class, object, methods, properties, constructor

Class: class is container that holds related methods, properties and constructor

- class act as a blue print for creating object

method: function inside a class is called method

function keyword is not needed to define method

properties: variables inside a class is called properties

var, let, const keywords are not needed to define properties

Object: instace of a class

- it is created from class

constructor- it is a function, used to initialize values to properties

this - this keyword is used to access methods and properties inside a class

this- called instace variable

this- refers to properties and methods inside that specific class

Syntax to create a Class

-----

```
class Class_name{
 properties
 constructor
 methods
}
```

- thses properties, constructor and methods are not mandatory, all are optional

eg:

```
class Student{
 name;
 age;
```

```

 constructor(){
 this.name= "sulthan",
 this.age= 28
 }
 printDetails(){
 console.log("Student Name:", this.name);
 console.log("Student Age:", this.age)
 }
}
const obj = new Student();
obj.printDetails()

```

- if constructor is expecting some arguments, then we need to pass these parameters while creating the object
- if any method inside the class expects some arguments, then pass the parameter while calling that method

## Pillars Of OOPS

-----

- 1) Inheritance
- 2) Polymorphism
- 3) Encapsulation
- 4) Abstraction

### 1) Inheritance:

- It is a mechanism that allows a class to inherit properties and methods from another class
- mainly child class can access methods and properties in parent class
- extends keyword is used to implement inheritance

### 2) Polymorphism

- behave differently based on number of arguments or method name from where it is calling

#### a) method overloading

- not supported in javascript
- consider there are two methods with same, first method is accepting no argument and second method is accepting one argument. If we call that method with no argument, it executes first method and if we call method with one argument, it executes second method
- but in javascript it calls the last method in every case

#### 2) method overriding:

consider we have a parent class and a child class. In both classes there are methods with same name  
 child class is inherited from parent class  
 if we create an object of child class and call any method that is present in both parent and child, method inside child class will execute  
 method inside child class overrides method in parent class

### 3) Abstarction:

- It is hiding the complex implemetation details of a system and exposing only the essential methods and features

eg:

Date class

### 4) Encapsulation:

- It provide privacy and maintain control over the transparency of properties and methods incide a class

- javascript not supporting access specifiers

For this Access specifiers are used

3 types of access specifiers

-----

1) Public

2) Private

3) Protected

1) Public:

- default access specifiers
- The class memeber can be accessible from any where

2) Protected:

- class memeber is accessible inside that class and its sub classes(child class)

3) Private:

- class member is accessible only within that class

### Javascript With HTML

-----

#### DOM (Document Object Model)

---

- while browser loads a HTML page, it create a tree like structure with the containing

html elements

- the tree consists of nodes, each node is an html element

#### How to apply javascript in HTML file

-----

1) internal

<script></script>

- it is better to use script after closing of body target

2) external js

<script src="url"></script>

#### How to access html elements

-----

1) by tag name

```
document.getElementsByTagName('tag_name')
```

- it returns collection
- collection is a data type created by browser, similar to array
- we have to convert this collection into a normal Array for using it

```
Array.from(collection_name)
```

- then map each item in the array and apply the style

eg:

```
Array.from(collection_name).map(item=>item.style.color="green")
```