JAVA SCRIPT

Introduction to JavaScript

- Javascript is used to create client side dynamic pages.
- Javascript is an object based scripting language.
- It is lightweight and cross platform.
- Javascript is a dynamically typed language.
- It doesn't require any compiler to compile the code in web browser it self has built in compiler.

What is JavaScript?

- Javascript is a lightweight object-oriented programming language used for client and server side technology.
- It was called scripting language before 2010 because it was used only for browser.
- But now we can use this language for database server, application server and mobile devices also so it is called programming language.

History of java script

- In 1993 mosaic was the first popular web browser to handle (GUI) but it could handle only static webpages.
- In 1994 Netscape corp started new project called Netscape navigator.
- Netscape navigator is a browser which have dynamic capabilities.
- It was invented by Brendan Eich. He was given the task to create programming language which would add dynamic capabilities to browser.
- It has been officially released on 1995 as name called mocha. Later they give name livescript.
- Due to marketing purpose they give name javascript. Because java was more popular during 1990's.

Current version of js

ES15 as per 2024

Features of JS

- All popular web browsers support javascript as they provide built in execution environments.
- Js follows the C language syntax and structure so it is called structured programming language.

- Js is an object oriented programming language like it has oops concept.
- It is a lightweight and interpreted language.
- It is a case sensitive language.
- Js is supported in several operating systems including windows, macos, linux etc.

Uses of JS

1. It is used to build simple and complex application such as google maps
2. It is used in game development
3. It is used in application development
4. It is used to build server
5. It is used to build client side application

Purpose of JS

- It is used to create dynamic interface on webpage
- It is used to do client side validation and server side validation
- It is used to interact or integration front end app and backend application

What kind of application we can build using JS

- We can build entire application
- We can build mobile apps like ios, android
- We can develop standalone application (desktop application like vs code, notepad, browser, ms paint etc.,)
- We can build machine learning application

JS ways to write code in HTML document

Javascript provides 3 ways to insert code in html document

1. **Head section**
   - We can write javascript code in body section by using script tag.
   - Inside script tag we can write javascript code

```
<head>
    <script>
     // js code in head section
</script>
<head>
```

2. **Body Section**
   - We can write javascript code in body section by using script tag.
   - Inside script tag we can write javascript code

<body>

<script>

// js code in body section

</script>

</body>

3. **External Js**
   - Here we will create separate .js file and connect to html document.
   - We can link js file in head section or body section by using script tag.
     Advantage: code reusability.

```
<head>
    <title>Document</title>
    <!-- linking external js file -->
    <script type="text/javascript"
src="external.js"></script>
</head>

<body>
    <!-- linking external js file -->    <script
type="text/javascript" src="external.js"></script>
</body>
```

Javascript comments

- The javascript comments are meaningful way to deliver message. It used add information about the code.
- To make code easy to understand.
- To avoid the unnecessary code.
- The javascript comment is ignored by the javascript engine i.e. embedded in the browser.

```
<script>

    // Javascript comments

    // In js we have two types of comments


    // 1. single line comments

    // console.log("hello world")
```

```
        // 2. block level comments
        /*console.log("hello world")
        console.log("hello how are you")*/
    </script>
```

Java Script outputs:

1. WE can print Js outputs in developer console with different

    i. **console.log()**
    It will print output in developer console as a message
    ii. **console.error()**
    It will print output text/content int red color as a error message
    iii. **console.warn()**
    It will print output as a warning message in yellow colour
    iv. **console.info()**
    It is same as consoling , it will print as a normal text

2. We can print JS outputs directly on webpage screen

    document.write()

    Example:

    Document.write("<h1> Welecome </h1>");

3. We can write JS output as alert/dialog box onn te webpage

    window.alert() or alert()

4. Java script inputs

    prompt or windowprompt()

Javascript Variables

- Variables are the storage locations for the data.
- variables are the references which points to memory location.

why do we require variables?
variables are required to store the data in the memory while executing the program, so that we can utilize the data to do some action or after executing the program we might required to store some output

Rules for naming variables

- Can include uppercase and lowercase letters, digits, (_), and ($) special characters.
- Can't begin with a digit.
- Keywords cannot be used as identifiers.
- Whitespaces cannot be used while naming identifiers.
- JS is case sensitive where var a and var A is different.

How to create a variable in Javascript??

We can create a variable in Javascript by using scoping statements or keywords, we can create a variables

- Var → old version JS
- Let → ES6 version JS
- Const → ES6 version JS

var name

let city

const age = 10

Variable declaration

var name

let city

Variable assignment (=)

name = 'virat'

city = 'Bangalore'

Variable initialization

const age = 30

var gender = 'male'

let a = 100

We have two different types of variables

1. **Local variable**
   This variable is declared inside the block and it can be accessed within the block only, not outside the block.

2. **Global variable**
   This variable is declared outside the block and it can be accessed by the any block.

Difference b/w var, let and const key word

1. var keyword

var can be redeclared and reassigned

2. let keyword

let can be reassigned and can’t be redeclared

3. const keyword

const can't be redecalared or reassigned

while decalaering only const should be intialized with value.

JavaScript Datatypes

- It is used to define the type of datatype which we are adding to variables.
- There are 8 different types of datatypes in JavaScript:

1. Number
2. String
3. Boolean
4. Null
5. Undefined
6. BigInt
7. Symbol
8. Object

All 8 different datatypes have been categorized into 2 types:

1. Primitive Datatype
2. Non Primitive Datatype

In JS, 95% of the time we use only Number, String, Boolean, and Object type datatypes.

Primitive Datatypes

- It is a basic datatype.
- The datatypes are immutable.
- It is stored in stack memory.

Primitive datatypes include:

1. Number
2. String
3. Boolean
4. Null
5. Undefined
6. BigInt
7. Symbol

Non Primitive Datatypes

- It is a complex datatype.
- The datatypes are mutable.
- It is stored in a heap memory.

Object type:

- Object
- Array

Primitive Datatypes

- **Number type:** It is a datatype of integers, decimal, and comes under number type.

var num = 27

let n = 90.8999

console.log(n)

// int and float does not support in js

- **String type:** Any data is enclosed with " or "" comes under string type.

var name = "virat"

let city = "mumbai"

const age = "35"

console.log(age)

- **Boolean type:** Any variables which contain value true or false, such type of data comes under Boolean type.

var isPlaced = true;

let isMarried = false;

- These are 3 datatypes + object datatype we use 95% in JS.
- In JS there are two special datatype values.

1. undefined

var a;

console.log(a);

2. null

let b = **null**;

console.log(b);

- **Bigint datatype:** Any value which is greater than 2 to the power of 53-1 comes under BigInt. Values less than it come under number type.

javascript

BigInt(1234567890123456789012345678901234567890123456789012345678901234567890n)

- **Symbol datatype:** Any data which is created using the Symbol() constructor comes under symbol types.

Non Primitive Datatypes

- **Object type:** Any variables—if we are assigning array or object—come under object type.

var person = {};

console.log(**typeof** person);

var names = [];

console.log(**typeof** names);

typeof operator

- In JS we have a special operator called typeof operator. It is used to find the datatype of any variable.
- **Syntax:** typeof <variable_name>

var id = 27

console.log(typeof id)

Operators in JavaScript

- JavaScript operators are symbols that are used to perform operations on operands.
- We have different types of operators. They are:

1. Arithmetic operators
2. Comparison operators
3. Bitwise operators
4. Logical operators
5. Assignment operators
6. Special operators

Extracted Text from Image

1. Arithmetic Operators

Operators	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

2. Comparision Operators

Operators	Description
= =	Is equal to
= = =	Identical
!=	Not equal to
! = =	Not identical
>	Greater than

Operators	Description
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

3. **Bitwise Operators Table**

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Bitwise Left Shift
>>	Bitwise Right Shift
>>>	Bitwise Right Shift 0

How Bitwise Operators Work

- Bitwise operators operate on **32-bit binary representations** of numbers.
- **&, |,** ^: Perform AND, OR, and XOR on each bit .
- ~: Inverts all bits; returns the two's complement.
- **<<, >>, >>>**: Shift bits left or right; **>>>** is unsigned and ignores the sign bit.

Usage

- Mainly used for **low-level tasks** such as performance flags, graphics, permissions, or binary data handling.

4. **Logical Operators Table**

Operator	**Description**
&&	Logical AND
||	Logical OR
!	Logical NOT

These operators are used for making decisions in conditions and control flow.

5. **Assignment Operators Table**

Operator	**Description**
=	Assign
+=	Add and Assign
-=	Subtract and Assign
*=	Multiply and Assign
/=	Divide and Assign
%=	Modulus and Assign

These operators update the value of a variable by performing the specified arithmetic operation and then assigning the result back to the variable.

6. **Special Operators Table**

Operator	Description
?:	Conditional operator returns value based on condition; works like if-else
,	Comma operator allows multiple expressions to be evaluated as a single statement
delete	Delete operator deletes a property from the object
in	In operator checks if object has the given property
new	Creates an instance
typeof	Checks the type of object

19/8/25

Functions in JavaScript

- Functions are the block of code which are defined to perform specific task.
- We call or reuse JavaScript functions n number of times. By many times we can call.

Advantages:

1. Code readability (Write once, use multiple times)
2. Less coding.

How to Create a Function in JavaScript

- When creating a function, the function name should represent a behaviour or name.
- The function is created by using the function keyword in JavaScript.
- Until and unless without invoking the function, the function wouldn't be executed.

Syntax:

text

```
function functionName() {
   // set of code
}
```

functionName(); // function call

Types of Functions in JavaScript

1. Named function
2. Parameterized function
3. Return function
4. Anonymous function
5. Callback function
6. Higher order function
7. Arrow function
8. IIFE's (Immediately Invoked Function Expressions)
9. Async function

1. Named Function

- This function is created by using the function keyword with function name. We will do coding.
- It is a normal function without parameters.

Example

```
function greet() {
   console.log("Hello welcome to JS code!");
}
greet();
```

Example

```
function add() {
   let x = 100;
   let y = 200;
   let z = x + y;
   console.log(z);
}
add(); // 300
```

2. Parameterized Function

- Passing parameters to function → function has input and output.
- The function which accepts parameters as input by taking those parameters and performing operation, finally giving output is called parameterized function.
- In one function, we can give n numbers of parameters.

Syntax

function functionName(parameters) {

// set of code

}

Example values:

- a = 10
- b = 20

functionName(arguments);

example: functionName(10, 20)

Parameters

- These are the variables without a reopening token, which can be used only inside the function block.

Arguments

- Arguments are the data passed to the parameters of the function.

3. Return Function

- Passing parameters → function → function call → output → returned as output.
- The function which returns the output, and that output is caught by separate variable, and then we can print the output.
- The return function can return string, boolean, number, or function objects.
- The return should be last executable statement in the function.

Syntax

function functionName() {

return;

}

4. Anynomous function

The function which is created by using function keyword without function name is called Anynomous function

Syntax:

```
function(){
        // set of code
}
```

Example:

```
let fun = function{
        console.log(“I am a anynomous “)
}
fun(); // I am a  anynomous
```

Function Expression

- **Definition:** Assigning a variable to unnamed functions and calling them using the variable name is called a function expression.
- **Example (Parameterized Anonymous Function):**

```
let mul = function(x, y) {
  let z = x * y;
  console.log(z);
}
mul(10, 2); // Output: 20
```

Return Anonymous Function

- Example:

```
let a = function() {
  return "Hello world";
}
console.log(a()); // Output: Hello world
```

5. Callback Function

- The function which we are passing as an argument to the another function is call callback function
- The anynomous and arro function are mainly used as callback function

6. Higher order Function

- The function which is taking another function as an argument those functions are called higher order function
- The function which is taking atleast one function as an argument is called Higher order function

7. Arrow Function

- The arrow function has been introduced ES6 feature of Java script
- The arrow is created by using arrow symbol without function name and function keyword
- Mainly arrow function is used as call back function

Syntax:

```
()=>{
        // set of code
}
```

Examples

- **Named Arrow Function**:

```
let fun = () => {
  console.log("I am ARROW function")
}
fun(); // Output: I am ARROW function
```

- **Parameterized Arrow Function**:

```
let add = (a, b) => {
  console.log(a + b);
}
add(30, 40); // Output: 70
```

- **Return Arrow Function**:

```
let num = () => {
  return 500;
}
console.log(num()); // Output: 500
```

- **Parameterized Returned Arrow Function (short form)**:

```
let yes = (a, b) => a + b;
yes(20, 50); // Output: 70
```

Arrow Function as Callback Example

- Shows function fn1 taking another function (including arrow functions or anonymous functions) as a parameter and invoking it.
- You can pass:
  - Anonymous function:

```
fn1(function () {
  console.log("Anonymous")
})
```

  - Parameterized arrow function:

```
fn1((a, b) => {
  return console.log(a + b)
})
```

8. IIFE Definition

- **IIFE**: Functions that are defined and executed immediately.
- Primarily used in emergency situations or when quick, one-time execution is needed.

Syntax

javascript

```
(function () {
  // set of code
})()
```

Examples

- **Normal IIFE**:

```
(function () {
  console.log("Welcome to JS class")
})()
// Output: Welcome to JS class
```

- **Parameterized IIFE**:

```
(function (name, age, city) {
  console.log("My name is " + name + " I am " + age + " years old I am from " + city)
})("Virat", 36, "Mumbai")
// Output: My name is Virat I am 36 years old I am from Mumbai
```

Threading Models

- **Java** supports both **single-threaded** and **multi-threaded** execution.
- **JavaScript (JS)** is primarily a **single-threaded language**, meaning it processes one command at a time.
- JavaScript can be made to behave like a multi-threaded language using certain features.

Synchronous vs Asynchronous

- **Synchronous (Sync)**: JavaScript executes code in a sequential, step-by-step manner.
- **Asynchronous (Async)**: Tasks that can run independently of the main program flow, such as:
    - **Network requests**
    - **Data fetch / API calling**
    - **Timers**
    - **Promises**
    - **async/await**

Notes

- JavaScript is fundamentally synchronous.
- Asynchronous features allow it to handle multi-threaded operations conceptually, even though true parallel threads are not used

9. async function

- Async functions introduced in es8 make it easier to work with promises. they are declared with the 'async' keyword and use the 'await' keyword to pause execution until promise is resolved
- Asynchronous function programming means multiple tasks can be executed concurrently and the program doesn’t wait for a task to finish before moving to the next one.

Difference between asynchronous function and synchronous function

Asynchronous function	**Synchronous function**
In asynchronous programming multiple task can be executed concurrently.	In synchronous programming the program execution occurs sequentially.

```
// asynchronous code
console.log("statement 1")
setTimeout(()=>{
console.log("statement 2")
},2000)
console.log("statement 3")
```

```
// synchronous code
console.log("statement 1")
console.log("statement 2")
console.log("statement 3")
```

Java script Objects

- JavaScript is object based language.
- An object is a combination of properties & behaviour; it refers to real world physical entity.
- Object is concept where we can store multiple data in a single variable.
- JavaScript is template based, not class based. We don’t use class to create a object in JavaScript.
- In JavaScript we can add properties & behaviour in the form of key value pairs that we will be calling properties inside object.
- In JS object all the key are unique keys.

How to create object in javascript?

- there are three ways to create object in javascript

1. By using object literals
2. By using instance of object
3. By using an object constructor

By using object literals

Syntax:
var object_name={key:value}
let
const

By using instance of object

Syntax:
var object_name=new Object()
let
const

By using an object constructor

- Here we need to create function with arguments each argument value can be assigned in the current object by using this keyword.
- The this keyword refers to the current object.

Crud operation in Object

1. read or access the data
2. insert or write the data
3. update the data
4. delete the data

1. how to access or read specific property from the object?

syntax:
object_name.keyname or
object_name[keyname]

2.how to modify or update value of a property from the object?

syntax:
object_name.keyname to update = new value

3.how to insert or add or write new property inside the object?

syntax:
object.keyname = newvalue or
object["keyname"] = newvalue

4.how to delete property or remove delete property from the object?

- **syntax:**
  delete object_name.keyname
- delete is a special key word in javascript

How to create a nested object in Javascript ???

```
Let Person = {
name: "virat"
id: 18,
age: 36,
address: {
city: 'Bangalore',
state: 'karnataka',
Pincode: 560007,
area: {
doorno: 114,
street: "1st main road",
location: "vijayanagar",
}
}
}

console.log(Person)
console.log(Person.name) // virat
console.log(Person.address.state);
// karnataka
console.log(Person.address.area.doorno)
// 114
```

How to add functions inside the Object???

```
let Person = {
name: 'virat',
age: 36,
city: 'mumbai',
play: function() {
console.log(this.name + " is playing")
},
run: function() {
console.log(this.name + " is running to " + this.city)
},
gender: "male"
}

console.log(Person.age); // 36
Person.play(); // virat is playing
Person.run(); // virat is running to Mumbai
```

Object Methods

How to retrive only keys from object?

syntax:
Object.keys(object_name)

How to retrive only values from object?

syntax:
object.values(object_name)

How to seal object?

Syntax:
Object.seal(object_name)

Object.seal(obj) seals an object means

- you cannot add or delete properties
- you can still modify the values of exisiting properties. Exisitng properties become non-configurable (can't be deleted or reconfigured)

What is Object.assign() method?

- Object.assign(target,...sources) copies the own enumerable properties from one or more source object to a target

26/08/25

JavaScript Arrays

1. Arrays are kind of data structure where we can store any type of data and there is no restriction of storing the data
2. Array is the indexed collection of data
3. In the array each data, we will call it as element
4. In the array we can store different types of datatypes

How to declare an array in JS?

In javascript we can declare an array in two different ways

1. Array Literals
2. Array Constructor

Array Literals

Syntax:

var array_name=[]

Example:

let arr = [10,29.9, true, ”A”, “Virat”, null, {name: “Sachin”, age:34}, undefined]

console.log(arr)

Array Constructor:

Syntax:

var array = new Array()

Example:

let num = new Array(10,20,30,40)

console.log(num)

o/p: [10,20,30,40]

How to access the element inside the array?

We can access the elements inside the array by using the index values

let arr = [‘Virat’, 777, true, {name: ‘Sachin’, city: ‘Mysore’}, false, “C”, 27.98]

console.log(arr)

console.log(arr[1]) //777

console.log(arr[3].city) // Mysore

console.log(arr[5]) // C

How to find the length of the given array

We can find the length of the given array by using length property

let arr1 = [‘Virat’, 777, true, {name: ‘Sachin’, city: ‘Mysore’}, false, “C”, 27.98]

console.log(arr1.length)

How to create 2D Array

Example:

let arr2=[10,20,'Virat',true,[40,50,60,{name:'Raj'}],false,'Z','Banglore']

console.log(arr2)

console.log(arr2[4][3])

Array Methods

Basic methods

1. push()
2. pop()
3. shift()
4. unshift()
5. indexOf()
6. lastindexOf
7. join()
8. reverese()
9. slice()
10. splice()
11. includes()

Advance Array methods

1. forEach()
2. math()
3. filter()
4. reduce()
5. sort()
6. toString()
7. delete()
8. isArray()
9. sum()

Basic Methods

push()

- The **push()** method appends data to the end of an array.

- **Syntax:** array.push(value).
- Each call to push() adds the value to the last position in the array.

pop()

- The **pop()** method removes the last element from an array.
- Each call to pop() deletes the element at the highest index (the end of the array).

unshift()

- The **unshift()** method adds new elements to the **beginning** of an array.
- Using unshift() shifts existing elements right and places the new value at index 0.

indexOf()

- this method is used to find the index value of particular element if the element is not avaiable inside the array then the indexOf() will return -1.

lastIndexOf()

- this method is used to find the last index value of particular element.

join()

- whenever we need to join all the elements inside the array join() should be used join() will join all the elements of an array and return it in the string format

includes()

- includes() checks weather the data passed to it is present inside the array or not. if its present then it returns true else it returns false

reverse()

- In order to reverse an array we use reverse()

slice()

- whenever we need to extract particular elements from array we can use slice() it will not alter the original array.
- **syntax:**
  slice(startindex,endindex)

splice()

- extraction starts at the start index extracts the length number of elements from the array in case splice() it also alters the original array
- **Syntax:**
  splice(start,length)

Advance Array Methods

forEach()

- if we want to access each and every element of an array we are using forEach(), forEach() will not return any value it will not alter original array.
- **syntax:**
  <array_name>.forEach(function(ele,index){
  console.log(element,index)
  })

map()

- map() should be return some value returned value always added in new array map() will not alter original array

filter()

- it is used to filter elements or data from the array based on certain condition filter callback function will return always boolean value it will not alter original array it will create new array
- **syntax:**
  <array_name>.filter(function(element, index){
  return <boolean value>
  })

reduce()

- It is used to reduce the array to one single value reduce method is such a method which accepts a callback function as the parameter (first param) (second param) initial value
- **syntax:**
  reduce(callback, initial_value)

sort()

- Its used to sort the array either in ascending or descending order

toString()

- the toString() returns a string with array values separated by commas and it does not change the original array

delete()

- array elements can be deleted using javascript operator delete using delete leaves undefined holes in the array

syntax: delete arrayname[index]

isArray()

- The isArray() method returns true if an variable storing an array otherwise false

syntax: Array.isArray(variable name)

Javascript Strings
It is a collection of characters enclosed within single or double quotes is called JS strings.

Let name = "Virat" // String
Let a = 'S'
Var isvalid = "true" // Valid

Let name = Raj // Invalid

String concatenation:
The process of combining two or more strings is called string concatenation.
Here we can combine the strings by using '+' operator.

How to find the length of a given string
Here we can find length of a string by using length property.

String Methods

1. charAt()
2. toLowerCase()
3. toUpperCase()
4. slice()
5. substring()
6. replace()
7. replaceAll()
8. trim()
9. concat()

charAt()

- whenever from a string if we need to extract a specific character we use charAt()

toLowerCase()

- all the characters of the string will be converted to lower case.

toUpperCase()

- all the characters of the string will be converted to upper case.

substring()

- its almost similar to slice(). the difference between substring and slice() is substring doesn't support -ve indexes

slice()

- when ever we want to extact a substring from a string we can use slice().
- **syntax:**

  slice(begin_index)
  slice(begin_index,end_index)

- It will accepts negative index it starts extraction from -2 index till the end and it ignores the value present at the index -2

replace()

- when we want to replace a particular string with new string we use replace method()

replaceAll()

- to replace all the occurances with new string we can't use replace() rather we should use replaceAll()

trim()

- In order to remove the whitespaces from the string we use trim()
- trim() will remove only whitespaces before and after the string and it won't remove the whitespaces present between the strings

concat()

- it is used to concat two string

selection statement or control statement

- we can control the execution of the program using control or selection statement: we can execute certain set of statements only if certain condition become true.

types of control statement

1. if
2. if else
3. else if
4. switch
5. ternary operator

If Condition

syntax

```
if(condition){
   some statement
}
```

If else Condition

syntax:

```
if(condition){
   //set of statement
}
else{
   //set of statement
}
```

else if Condition

syntax:

```
if(condition){
   //set of statements
}
else if(condition){
   //set of statements
}
else if(condition){
   //set of statements
}
else if(condition){
   //set of statements
}
```

Switch Condition

syntax:

```
switch(){
   case "value1":{
```

```
        //set of statement
        //break;
    }
    case "value":{
        //set of statement
        //break;
    }
    default:{
        //set of statement
    }
}
```

ternary operator

syntax:

condition ? //true statement : //false statement

Looping Statements in Javascript

Loops are used to complete repatative tasks instead of manually repeateing the tasks
We must take advantages of loops

We have different types of loops in Javascript.

1. for loop
2. while loop
3. do while loop
4. for in loop
5. for of loop(ES6 version of Javascript)

for loop:

Syntax:

```
for(initialize; condition; increment/decrement)
{
  // set of statement.
}
```

while loop:

Syntax:

```
while(condition){
  // set of code
  increment/decrement.
}
```

do while loop

syntax:

```
do{
   //execute the statements
}
while(condition)
```

for in loop

- we can iterate through the object and we can get the access of all keys of object properties mainly this loop will access the key of the object.

syntax:

```
for(<variable declaration> in <object name>){
}
```

for of loop

- we can iterate through the array and we can access of all the elements in an array.

syntax:

```
for(variable_name of array_name){
}
```

NAN operator

- In javascript NAN stands for "not a number" and it is special value that represents an invalid number

when does NAN occur?

- when you perform mathematical operation that doesn't return a valid number

```
let result = 0/0          // NaN

let c = 0 + 0             // 0

let b = 100 / "S"         // NaN

const value = Math.sqrt(-1)       // NaN

let num = parseInt("5")           // 5

let num1 = parseInt("abc")        // NaN


let a = 9

console.log(typeof a)     // number

console.log(typeof NaN)   // number

NaN === NaN               // false
```

DOM (Document Object Model)

Dom is a interface between Javascript and the browser.
In order Javascript wants to communicate with browser, it takes help of dom.

Dom is a tree (like structure)

```
<html>
  <head>
    <title></title>
    <link>
    <script>
  </head>
  <body>
    <h1></h1>
    <p></p>
    <div>
      <h3>
      <img>
      <input type="text" password>
```

```
  </div>
 </body>
</html>
```

[Tree Structure Diagram:]

```
Document
└── html
    ├── head
    │   ├── title
    │   ├── link
    │   └── script
    └── body
        ├── p
        ├── div
        │   ├── h3
        │   ├── img
        │   └── input (type="text", password)
        └── styling
```

main uses of dom

1. allow us to make js interact with browser
2. we can write js to create, modify and delete html elements
3. we can dynamically add styles, classes and attributes to the html elements
4. dom is very complex api (application programming interface) which contains lots of methods and properties which help us to interact with browser

why dom is created?

- using javascript we can manipulate the html document so that we can create dynamic user interface

dom manipulation

- how to access dom node (html element)
- how to add new html element
- how to remove existing html element

- how to change content or attributes of html elements
- to perform above operation browser is providing us predefined objects and functions called dom api

Dom methods

1. document.write()
2. document.head
3. document.body
4. document.url
5. document.getElementById()
6. document.getElementsByClassName()
7. document.getElementsByTagName()
8. document.getElementsByName()
9. document.querySelector()
10. document.querySelectorAll()
11. document.createElement()
12. remove()
13. getAttribute()
14. setAttribute()
15. document.title

document.write():

- it helps us write some text directly to the html document.

document.head

- it helps to access entire head section of the html document

document.body:

- it helps to access entire body section of the html document

document.url

- it helps to access url address of the html document

document.title

- it helps to access the title of the webpage

document.getElementById()

- its used to find the elements on the webpage based on the Id attribute. it will return single element which matches with id value.

document.getElementByName()

- its used to find the elements on the webpage based on the name attribute, it returns nodelist it is nothing but array of node or html elements

document.getElementByTagName()

- Its used to find the elements using the html tagname

document.getElementByClassName()

- Its used to find the elements using the html class name
- It will return the output in nodelist format or array format

document.querySelector()

- querySelector() returns the first element in the document that matches the specific selector or group of selector
- if no matches found then it will return null

document.createElement()

- we can create web elements dynamically using createElement()
- In order to dynamically append created element to weppage we can take the help of appendChild() or append()

remove()

- By using remove() we can remove web elements dynamically from html document

setAttribute()

- The setAttribute(name,value) method is used to set a new attribute or change the attribute value of existing attribute on an html element

getAttribute()

- the getAttribute(name) method retrieves the value of a specific attribute

Java Script Events

- All actions in js we call it as event
- To find how many events we have in js type in console onclick
- events----->actions
- event is nothing but some activity on the webpage.
- event consits of two things eventlistener and event handler

Eventlistener

- It is used to catch the event occured on the Element
- **Example:** onclick, onchange, onload, onsubmit, onkeydown, onmouseover, onmouseout

Eventhandler

- It is function or js function call
  <button onclick="fn()">click me</button>
- event is a predefined object in javascript where it contains
- all the information related to click event

onchange()

- onchange() events get called upon changing of dropdown

onmouseover and onmouseout

these are the events related to mouse move activity

onmouseover: when we take the cursor on any web element it gets triggered

onmouseout: when we take out the cursor from any web element it gets triggered

onsubmit()

This is the event handler is used in form submission to do form validation

onfocus

the focus event listener in javascript is triggered when an element gains focus - usually through keyboard navigation (like pressing tab button) or when clicked

Adding event through javascript

We can listen to events through javascript by using addEventListener()

addEventListener() accepts 2 parameters

1. type of event(string) ex: click, change etc

2. callback function -->

Math Object

Math is an inbuilt object in javascript which can be used to perform some specific math operation

1. Math.PI
2. Math.sqrt
3. Math.pow
4. Math.round

5. Math.floor
6. Math.trunc
7. Math,sign
8. Math,random

Date Object

- When ever we need to know about the date we should use date
- object gives information regarding Day, Month, Year, Minutes, Hours, Seconds, and Time Zone
- months--->0-->jan 11-->dec
- week--->0-->sunday 6-->Saturday

1. getDate()
2. getMonth()
3. getDay()
4. getFullYear()
5. getHours()
6. getMinutes()
7. getSeconds()

Timers

Whenever we wanted to delay the execution or we need to call the function again and again we use timers

Timers are two types

1. setTimeout()
2. setInterval()

what is the difference between setTimeout() and setInterval()

the main difference between the setTimeout() and setInterval (), the setTimeOut() will execute the callback() only once where as setInterval() keeps on triggering the callback() regularly after the given interval of time(unless you stop it)

Hoisting

Hoisitting is js default behavior of moving the declerations to the top of the current scope (either the global scope or a function scope) during the compliation phase before code execution

-->only declarations are hosited, not initialization /assignments

-->both variables and function are hoisted but behave differently

How it works internally?

when js engine starts

1. first phase (memory creation phase)

--> it scans the code

--> it creates memory space for variables and

----> variable declarations are set to undefined

----> function declaration are fully hosited (entire function definition)

2. second phase

code runs line by line

Hoisting for var

```
//Hositing for var
// console.log(a) //undefined
// var a=10
// console.log(a) //10
// var b //hosited
// console.log(b) //undefined
//hositing for let and const
let b=20
console.log(b) //20
console.log(c) //reference error
let c=200
//let and const are also hosited but kept in //(TDZ) temporal dead zone a period where the
//variable exisits but is not accessible until //its initalized
let d; //hosited
console.log(d) //undefined
d=900
console.log(d) //900
//Hoisting for function
greet() //good morning
function greet(){
```

```
   console.log("good morning")
}

//entire function greet is hoisted
//so you can call it before declaration

//note: only function declarations are fully hoisted function expressions and arrow
//functions are behave like variables
sayHello() //type error
var sayHello() = () =>{
        console.log(“hello world”)
}
sayHello()//helloworld
```

Clousers in javascript

It is a technique to access of outer function variable inside inner function (or)

It is function to define function inside another function

A clouser can be defined as a javascript feature in which the inner function has access to the outerfunction variable, javascript, every time a closure is created with the creation of function

The clousers has three scope claims listed as follows

1. access to its own scope
2. access to the variable of the outer function
3. access to the global variables

syntax:

```
// function outer(){
//   ----------
//   ----------
//   ----------
//   function inner(){
//     ----------
```

```
//     ----------
//     ----------
//   }
//   return inner
// }
```

Destructing of Array or Object

1) It is one of the ES6 feature!

2) It is a technique in Java script used to unpack the properties or elements from the object or Array and assigning to individual variables is called destructing.

```
//Old approach
let name = person.name
let age = person.age
let city = person.city
console.log(name)  //virat
console.log(age)   //36
console.log(city)  //mumbai


//New approach
const {name, age, city, gender} = person
console.log(city)   //mumbai
console.log(gender) //male
```

Spread operator

...<object name>

or

...<Array name>

it is used to copy properties of one object into another object and it is used to copy elements of one array into another array

example: applying spread operator for object

Asynchronous programming

Asynchronous programming is a programming paradigm that allows tasks to be executed concurrently, without blocking the main thread of execution. This approach is particularly useful when dealing with operations that might take time to complete, such as network requests, file I/O, or timers, allowing the program to continue running other tasks while waiting for the operation to complete.

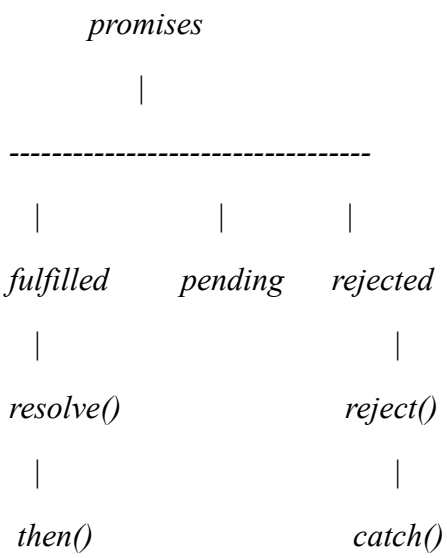
Asynchronous programming can be done by using timers, /promises, async and await

Promise in javascript

- These are the way to handle the asynchronous operations
- A promise is a javascript object that links producing and consuming code that represent a value may be available now or in the future or never

The promise object can be in any 3 states

1. pending
2. fulfilled
3. rejected



//creating a promise

//create a promise using promise constructor take a function

//with two parameter resolve and reject

//here the promise will be either be resolved or it will be rejected

```
let promDemo = new Promise((resolve, reject) => {
  if(true){
```

```
    resolve('Promise Fulfilled')
  }
  else{
    reject("Promise Rejected")
  }
})

//fetch()

//Handling promise result

//we can use .then() method to handle the promise result
//if any error occurs, then we use .catch() method to handle the result
promDemo.then((result) => {
  console.log(result)
}).catch((error) => {
  console.log(error)
})
```

Fetch:

- fetch is a method which works internally as a promise. fetch method is used to make a network request or to call a API in js
- Fetch will return a promise in js

async and await

async: declare a function or method as asynchronous and can

pause its execution to wait for completion of other process

await: make a suspension point where execution may wait for

the result of async function or methods

Template literals

- Template literals provide an easy way to interpolate variables and expressions into strings.
- template literals use back-ticks (``) rather than the quotes ("") to define string

Rest Parameters

Rest Parameters allow a function to accept an indefinite number of arguments as an array and also it can applied for array or object to store elements or properties

syntax:

```
function myfun(...args){
   console.log(args)
}
```

This keyword

This is a special keyword in js which point to the current value of the object

We can change the reference of 'this' keyword using three predefined functions. They are

1. call
2. apply
3. bind

call( )

For this function first argument should be always object name

For which this keyword is point/is referred second argument is function accepting argument

apply( )

The apply function it will take two arguments first argument is object and second argument is array

We can give function arguments inside the array

PROJECTS

1. Background Colour Change
2. Calculator
3. QR Code Generator