# JAVA SCRIPT

Java script is one of the most popular scripting language /popular programming language/object based programming language.

Java script is used in millions of website to improve design validate the form detect the browser and many more.

More than 80% of the website use the JavaScript for client side scripting.

Client side scripting means the JavaScript code will compile on the web browser/client system.

All the browser will have a default JavaScript compiler.

* chrome: V8 engine.
* safari: JavaScriptCore

### Why JavaScript:

* It is used to add functionality and interactivity to the web page.
* It is a light weight and object based programming language.
* Java Script can be directly embedded to HTML page.
* JavaScript is an interpreted language.
* It is an open-source language.
* JavaScript is case sensitive language.

### JavaScript was invented by Brendan Eich in 1995.

In html we can add the JavaScript using the script tag in head tag or the body tag.

### <script>…</script>

Or

we can create a separate file and add the JavaScript code and add the link to the html document with script tag.

Eg:

### <script src="index.js"></script>

**JavaScript Variables**

Variables are "containers" for storing information. JavaScript variables are used to hold values or expressions.

### Rules for JavaScript variable names:

* Variable names are case sensitive (y and Y are two different variables)
* Variable names must begin with a letter or the underscore character **Note:** Because JavaScript is case-sensitive, variable names are case-sensitive. **Syntax:**

### Keyword variable\_name=value; Eg:

var a=10; let b=10; const c=20;

### declaring a variable in js.

Declaring the variable means creating the variable but the value for the variable is not assigned.

Eg:

var a; let b;

Assigning the value to the variable

After declaring the variable, the value of the value can be assigned. a=10;

b=20;

### Redeclaring JavaScript Variables

If you redeclare a JavaScript variable, it will not lose its original value. var x=5;

x;

After the execution of the statements above, the variable x will still have the value of 5. The value of x is not reset (or cleared) when you redeclare it.

### Data Types in javascript. Number

The numbers are values that can be processed and calculated. You don't enclose them in quotation marks. The numbers can be either positive or negative floating and decimal numbers.

Eg:

var a=10;

The datatype of **a** is Number. let a=20.34

the datatype of **b** is Number.

### Strings

String is a series of letters and numbers enclosed in quotation marks. Eg:

let name="Suhas" let b;

b="good morning"

### Boolean (true/false)

let’s you evaluate whether a condition meets or does not meet specified criteria. let isLoggin=false;

var accept=true;

### Null

Null is an empty value. null is not the same as 0 it is calculable number, whereas null is the absence of any value.

let a=null; console.log(a) **Undefined**

The undefined represents the value is not defined but at later time the value will be assigned.

let a; console.log(a)

### Operators in JavaScript

**JavaScript Arithmetic Operators**

Arithmetic operators are used to perform arithmetic between variables and/or values. Given that y=5, the table below explains the arithmetic operators:

|  |  |  |  |
| --- | --- | --- | --- |
| **OPERATOR** | **DESCRIPTION** | **EXAMPLE** | **RESULT** |
| + | Addition | X = y + x | X = 7 |
| - | Subtraction | X = y – x | X = 3 |
| \* | Multiplication | X = y \* x | X = 10 |
| / | Division | X = y / x | X = 2.5 |

|  |  |  |  |
| --- | --- | --- | --- |
| % | Modulus | X = y % x | X = 1 |
| ++ | Increment | X = ++y | X = 6 |
| -- | Decrement | X = --y | X = 4 |

### JavaScript Assignment Operators

Assignment operators are used to assign values to JavaScript variables. Given that x=10 and y=5, the table below explains the assignment operators:

|  |  |  |  |
| --- | --- | --- | --- |
| **OPERATOR** | **EXAMPLE** | **SAME AS** | **RESULT** |
| = | X = Y | X = Y | X = 5 |
| + = | X + = Y | X = X + Y | X = 15 |
| - = | X - = Y | X = X – Y | X = 5 |
| \* = | X \* = Y | X = X \* Y | X = 50 |
| / = | X / = Y | X = X / Y | X = 2 |
| % = | X % = Y | X = X % Y | X = 0 |

### Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that x=5, the table below explains the comparison operators:

|  |  |  |
| --- | --- | --- |
| **OPERATOR** | **DESCRIPTION** | **EXAMPLE** |
| == | Is equal to | x==8 is false |
| === | Is exactly equal to (value and type) | x===5 is true, x==="5" is false |
| != | Is not equal to | x!=8 is true |
| > | Greater than | x>8 is false |
| < | Less than | x<8 is true |

|  |  |  |
| --- | --- | --- |
| >= | Greater than equal to | x>=8 is false |
| <= | Less than equal to | x<=8 is true |

### Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that x=6 and y=3, the table below explains the logical operators:

|  |  |  |
| --- | --- | --- |
| **OPERATOR** | **DESCRIPTION** | **EXAMPLE** |
| && | And | (x < 10 && y > 1) is true |
| || | Or | (x==5 || y==5) is false |
| ! | not | !(x==y) is true |

### String concatenation

String concatenation means adding two or more string to one string The + operator is used to concatenate two strings in JavaScript.

let fname="Suhas " let lname="S J"

let name=fname+lname console.log(name) output:

Suhas S J

### Adding Strings and Numbers

let x=5+"5"

console.log(x) output is 55

when you try to concatenate a string and the number then the number is explicitly convert to string and then the output will be 55.

let x=5+5

console.log(x) output is 10

when you try to add 2 numbers then there will be no type casting so that the output will be 10.

### Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

* if statement - use this statement if you want to execute some code only if a specified condition is true
* if...else statement - use this statement if you want to execute some code if the condition is true and another code if the condition is false
* if...else if else statement - use this statement if you want to select one of many blocks

of code to be executed

* switch statement - use this statement if you want to select one of many blocks of code to be executed

### If Statement

You should use the if statement if you want to execute some code only if a specified condition is true.

### Syntax

If(condition)

{

Code to execute

}

Note that if is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!

Eg:

let age=18 if(age<18) {

console.log(“not eligible for DL”)

}

### If...else Statement

If you want to execute some code if a condition is true and another code if the condition is not true, use the if else statement.

If(condition)

{

Code to execute

}

else {

other code execute

}

Eg:

let age=18 if(age<18) {

console.log (“not eligible for DL”)

}

else {

console.log (“eligible for DL”)

}

### If...else if else Statement

You should use the if.... else if else statement if you want to select one of many sets

of lines to execute. If(condition)

{

Code to execute

}

else if {

some other code executes

}

else { other code

}

Eg:

let age=18 if(age<18) {

console.log (“age should be greater than 18”)

}

else if (age>60) {

console.log (“age less than 60”)

}

else {

console.log (“eligible for DL”)

}

### The JavaScript Switch Statement

You should use the switch statement if you want to select one of many blocks of code to be executed.

Eg:

let a=5 switch(a) { case 1:

console.log ("1 is executed") break

case 2:

console.log ("2 is executed") break

case 3:

console.log ("3 is executed")

break case 4:

console.log ("4 is executed") break

case 5:

console.log ("5 is executed") break

case 6:

console.log ("6 is executed") break

}

Output:

5 is executed

### JavaScript Loops

Very often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In JavaScript there are two different kind of loops:

* for - loops through a block of code a specified number of times
* while - loops through a block of code while a specified condition is true

### The for Loop -

The for loop is used when you know in advance how many times the script should run. for (initialization; condition; increment/decrement)

{

Statement to execute

}

Eg:

for (var i = 0; i <= 10; i++) { console.log(i)

}

Output:

0

1

2

3

4

5

6

7

8

9

10

### The while loop

The while loop is used when you want the loop to execute and continue executing while the specified condition is true.

while (var<=end value)

{

// code to be executed

}

### Eg:

var i=0; while (i<=10)

{

document.write("The number is " + i); document.write("<br />");

i=i+1;

}

Output:

The number is 0 The number is 1 The number is 2 The number is 3 The number is 4 The number is 5 The number is 6 The number is 7 The number is 8 The number is 9 The number is 10

### The do...while Loop

This loop will always execute a block of code once, and then it will repeat the loop as long as the specified condition is true. This loop will always be executed at least once, even if the condition is false, because the code is executed before the condition is tested.

do

{

//code to be executed

}

while (var<=end value);

**Example** var i=0; do

{

document.write("The number is " + i); document.write("<br />");

i=i+1;

}

while (i<0);

### Result:

The number is 0

### JavaScript Break and Continue Statements

There are two special statements that can be used inside loops: break and continue.

### Break

The break command will break the loop and continue executing the code that follows after the loop (if any condition).

Example var i=0;

for (i=0;i<=10;i++)

{

if (i==3)

{

break;

}

document.write("The number is " + i); document.write("<br />");

}

### Result

The number is 0 The number is 1 The number is 2

### Continue

The continue command will break the current loop and continue with the next value.

### Example

var i=0

for (i=0;i<=10;i++)

{

if (i==3)

{

continue;

}

document.write("The number is " + i); document.write("<br />");

}

### Result

The number is 0 The number is 1 The number is 2 The number is 4 The number is 5 The number is 6 The number is 7 The number is 8 The number is 9 The number is 10

**JavaScript Functions**

A function is a self-contained piece of code that performs a particular task. You can recognize a function by its format - it's a piece of descriptive text, followed by open and close brackets. A function is a reusable code-block that will be executed by an event, or when the function is called.

## The syntax for creating a function is:

function function\_name(var1, var2..., varX)

{

some code

}

var1, var2, etc. are variables or values passed into the function. The {and the} defines the start and end of the function.

Note: A function with no parameters must include the parentheses () after the function name: function function\_name()

{

some code

}

Note: Do not forget about the importance of capitals in JavaScript! The word function must be written in lowercase letters, otherwise a JavaScript error occurs! Also note that you must call a function with the exact same capitals as in the function name.

## The return Statement

The return statement is used to specify the value that is returned from the function. So, functions that are going to return a value must use the return statement.

Example

The function below should return the product of two numbers (a and b): function prod(a,b)

{

x=a\*b; return x;

}

When you call the function above, you must pass along two parameters: product=prod (2,3);

The returned value from the prod () function is 6, and it will be stored in the variable called product.

### The Lifetime of JavaScript Variables

* When you declare a variable within a function, the variable can only be accessed within that function.
* When you exit the function, the variable is destroyed. These variables are called local variables.
* You can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared.
* If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed.

## Arrays in Java Script

An array is a special variable, which can hold more than one value

In Java Script arrays are heterogeneous means they can hold any type of variable.

In Java Script arrays are dynamic in nature which means they can be dynamically updated the value stored in java script can be easily updated at any time.

Ways of creating an array **Using an array literal Syntax:**

const array\_name = [item1, item2, ...];

Example

const array = [1, "suhas", true];

You can also create an array, and then provide the elements:

const array=[] array[0]=1; array[1]= "suhas" array[3]=true;

## using new Keyword

const array=new Array(1,"suhas”, true)

**Note**: Array indexes always start with 0.

## JavaScript Array Methods

**length:**

The length property returns the length (size) of an array. Ex:

const array=[10,20,11,23,42,87,34] console.log(array.length) //output:7 **toString():**

The JavaScript method toString() converts an array to a string of (comma separated) array

values. Eg:

const array=[10,20,11,23,42,87,34] let a=array.toString() console.log(a)

output:

10,20,11,23,42,87,34

## join():

The join() method also joins all array elements into a string.

It behaves just like toString(), but in addition you can specify the separator. const array=[10,20,11,23,42,87,34]

let a=array.join(";") console.log(a) Output:

10;20;11;23;42;87;34

## pop():

The pop() method removes the last element from an array. The pop() method **returns** the value that was **popped out**. Eg:

const array=[10,20,11,23,42,87,34] let a=array.pop()

console.log(a) output:

34

## Push():

The push() method adds a new element to an array at the end. The push() method **returns** the new length of array.

Eg:

const array=[10,20,11,23,42,87,34] let a=array.push(100) console.log(a)

output:

8

## Shift():

The shift() method removes the first array element and "shifts" all other elements to a lower index.

The shift() method returns the value that was removed. Eg:

const array=[10,20,11,23,42,87,34] let a=array.shift()

console.log(a) output:

10

## unshift():

The unshift() method add the element at first index. The unshift() method returns the new array length. Eg:

const array=[10,20,11,23,42,87,34] let a=array.shift()

console.log(a) output:

8

## Slice()

The slice(start index, end index) method slices out a piece of an array into a new array.

The slice() method creates a new array and does not remove any elements from the source array.

Start index included

End index excluded(end-1) Eg:

const array=[10,20,11,23,42,87,34] let a=array.slice(2,6)

output: 11,23,42,87

## JavaScript Object:

Objects are used to store the data in key-value pair. const person = {

firstName : "Suhas", lastName : "S J", age : 23,

height : 5.7

};

## Accessing JavaScript Properties

Syntax:

objectName.property Eg:

Console.log(person.age)

objectName["property"]

Eg:

Console.log(penson[age])

## Adding New Properties

You can add new properties to an existing object by simply giving it a value. If property is already existing, then that property will be overridden. person.nationality = "India";

## Deleting Properties

The delete keyword deletes a property from an object. const person = {

firstName : "Suhas", lastName : "S J", age : 23,

height : 5.7

};

delete person.age

The delete keyword deletes both the value of the property and the property itself.

## JavaScript ArrayOfObjects

We can store an objects inside the array. let obj=[{

name:'Suhas S J', age:23, place:'Bangalore', Height:5.7

},{

name:'Abhi ', age:24, place:'Delhi', Height:5.9

},{

name:'raj', age:29, place:'Bangalore', Height:5.7

},{

name:'pratheek', age:20, place:'Bangalore', Height:6.0

}]

We can acess the individual objects according to the indexes.

We can acess the object property using the index of object and key of object. Console.log(obj[0])//print object at 0 index.

Console.log(obj[0].name)//print the value of name of object at 0 index and key name.

## JavaScript String methods Length:

The length property returns the length of a string. Let name= "Suhas S J" Console.log(name.length)

## charAt()

The charAt() method returns the character at a specified index in a string. Let name= "Suhas S J"

Console.log(name.charAt(4))

charCodeAt()

The charCodeAt() method returns the code of the character at a specified index in a string. Let name= "Suhas S J"

Console.log(name.charCodeAt(4))

## slice()

slice() extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters

* start position included
* end position excluded

Let name= "Suhas S J" Console.log(name.slice(2,7) slice () will take negative index. Let name= "Suhas S J" Console.log(name.slice(-6,-1))

## substring()

substring() is similar to slice().

The difference is that start and end values will not take negative index it will be treated as 0. Let name= "Suhas S J"

Console.log(name.substring(2,7))

## substr()

substr() is similar to slice().

The difference is that the second parameter specifies the length of the extracted part Let name= "Suhas S J"

Console.log(name.substr(2,4))

**toUpperCase()** convert text to upper case. Let name= "Suhas S J"

Console.log(name.toUpperCase())

**toLowerCase()** convert text to lower case. Let name= "Suhas S J"

Console.log(name.toLowerCase())

## trim()

The trim() method removes whitespace from both sides of a string. Let name= " Suhas S J "

Console.log(name.trim())

## trimStart()

The trimStart() method will removes whitespace only from the start of a string. Let name= " Suhas S J "

Console.log(name.trimStart())

## trimEnd()

The trimEnd() method will removes whitespace only from the Ending of a string.

Let name= " Suhas S J " Console.log(name.trimEnd())

## padStart()

The padStart() method pads a string from the start. Accept 2 parameter

1. max length of the string
2. string to add Eg:

let a="10" console.log(a.padStart(3, "0"))

## padEnd()

The padEnt() method pads a string from the End. Accept 2 parameter

1. max length of the string
2. string to add Eg:

let a="10" console.log(a.padEnd(3, "0"))

## replace()

This method is used to replace a string with another String. let name="Suhas" console.log(name.replace("Suhas","Suhas S J"))

output:

Suhas S J

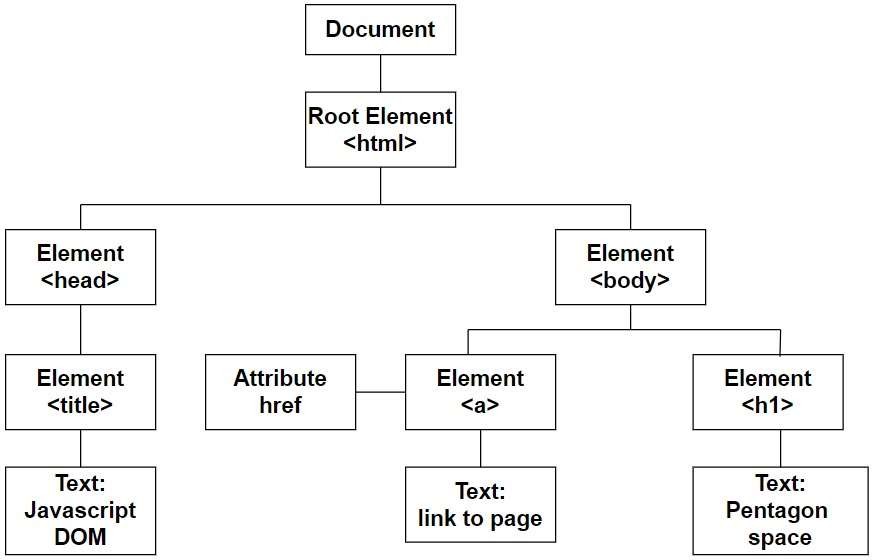
# DOM

DOM Abbreviates to **document object model**

When a web page is loaded, the browser creates a Document Object Model of the page.

The HTML DOM model is constructed as a tree of Objects with the **document** as the parent element.

With the help of DOM, we can access and manipulate the html element in JavaScript.



## DOM methods for accessing the element

* accessing HTML elements by id
* accessing HTML elements by tag name
* accessing HTML elements by class name
* accessing HTML elements by CSS selectors

## accessing HTML elements by id

Here we will acess the element by id name.

let element=document.getElementById("id\_name")

If the element is found, the method will return the element as an object. If the element is not found, myElement will contain null.

Eg:

<div id="container"></div>

let element=document.getElementById('container')

## accessing HTML elements by tag name

var elem = document.getElementsByTagName("h1") Here we will acess the element by Tag name.

Eg:

<div id="container"></div>

let element=document.getElementsByTagName('div')

## accessing HTML elements by class name

If you want to find all HTML elements with the same class name, use getElementsByClassName()

var elem = document.getElementsByClassName("main"); Eg:

<div id="container" class="divs"></div>

let element=document.getElementsByClassName('divs')

## accessing HTML elements by CSS selectors

If you want to find all HTML elements that matches a specified CSS selector (id, class names, types,), use the querySelector() method

var elem = document.querySelector(".main") Eg:

<div id="container" class="divs"></div>

let element=document. querySelector('.divs')

## createElement:

This method is used to create an html element. Eg:

const heading= document.createElement('h1')

## appendChild:

This method is used to add the element created to html document. document.body.appendChild(heading);

*Here the element will be added to the body of page*

let heading=document.createElement('h1')

document.getElementById('container').appendChild(heading)

*Here the element will be added in between the element with the id* ***container.***

## innerHTML:

it is used to add the html content inside the element. document.getElementById('container').innerHTML='<h1>Heading 1 created in DOM</h1>'

## innerText:

it is used to add the text content inside the element. document.getElementById('container').innerText='Text added in dom'

To add the style for the element: document.getElementById('container').style.backgroundColor='blue'

here we need to write the properties in camelCaseConvention and the value need to be specified in string format.

## remove:

It is used to remove an element from the document. let heading=document.createElement('h1') heading.remove()

## setAttribute:

It is used to add an attribute for the element. let heading=document.createElement('h1') heading.setAttribute('class','h1')

let heading=document.createElement('h1') heading.setAttribute('id','headings')

### removeAttribute:

It is used to remove the attribute from the element. let div=document.getElementById('container') div.removeAttribute('id')

***programme to take input from the user and print the sum in html document using DOM***

<!DOCTYPE html>

<html>

<head>

<title>DOM DEMO</title>

</head>

<body>

<p> The Sum is <span id="sum"> </span></p>

<script>

let num1=Number(prompt("Enter Value of number 1")) let num2=Number(prompt("Enter Value of number 2"))

let sum=num1+num2

document.getElementById('sum').innerText=sum

</script>

</body>

</html>

# Events in JavaScript

Event are the action or occurrence that happen in the web browser such as click, keypress, form submission, mouse hover.

JavaScript provide a build in mechanism for handling the events, allow you to create an interactive web application.

|  |  |
| --- | --- |
| **Event** | **Event Handler** |
| click | onclick |
| mouseover | onmouseover |
| mouseout | onmouseout |

|  |  |
| --- | --- |
| mousedown | onmousedown |
| keyup | onkeyup |
| keydown | onkeydown |
| Focus | onfocus |
| Submit | onsubmit |
| onload | onload |

EventLitesner:

This function that wait for the event to occur and response to it.

Event listener listen for the event and response for the event by calling a function. addEventListener():

This function is an example of higher order function which will take event as one argument as a string and a callback function which will affect when the event occurs.

## Inline Event Handlers.

Event handlers can directly add as an attribute to the html element . Eg:

1)

<!DOCTYPE html>

<html>

<head>

<title>DOM DEMO</title>

</head>

<body>

<button **onclick="toggle()"**>Click Here</button>

<script>

function toggle(){

document.write('Button Clicked By User')

}

</script>

</body>

</html> 2)

<!DOCTYPE html>

<html>

<head>

<title>DOM DEMO</title>

</head>

<body onload="loaded()">

<button>Click Here</button>

<script>

function loaded(){ alert('page loaded')

}

</script>

</body>

</html>

## DOM Event Handler:

Accessing the element using DOM methods and adding the events using the addEventListener method.

Eg:

<!DOCTYPE html>

<html>

<head>

<title>DOM DEMO</title>

</head>

<body>

<button id="btn">Click Here</button>

<script>

let btn=document.getElementById('btn') **btn.addEventListener('click’, function(){ alert('Button Clicked')**

### })

</script>

</body>

</html>

## Form Validation using DOM and Events

<!DOCTYPE html>

<html>

<head>

<title> JS EVENTS</title>

</head>

<body>

<form id="myform">

<input type="email" id="email" placeholder="Enter Email"><br>

<input type="Password" id="pwd" placeholder="Enter Password"><br>

<input type="Password" id="cpwd" placeholder="Enter Conform Password"><br>

<input type="button" value="Submit" onclick="validateForm()">

</form>

<script>

function validateForm()

{

let email=document.getElementById('email').value; let password=document.getElementById('pwd').value;

let password1=document.getElementById('cpwd').value;

if(email==''||password==''){

alert('email or password cannot be null')

}

else if(password!=password1){ alert('password mismatch')

}

else{

alert('data saved successfully') document.getElementById('email').value=''

document.getElementById('pwd').value='' document.getElementById('cpwd').value=''

}

}

</script>

</body>

</html>

## This keyword

In JavaScript, the this keyword refers to current object. Eg:

<!DOCTYPE html>

<html>

<head>

<title>this demo</title>

</head>

<body>

<script>let persion={ fname:'suhas', lname:'s j',

age:'22', place:'shivamogga', detail:function(){

document.write("Name is "+fname+" "+lname+"age is "+age+" from"+" "+place)

}

}

persion.detail()

</script>

</body>

</html>

Above code will result in error

To resolve the error we need to use this keyword Bellow is code for same

let persion={ fname:'suhas', lname:'s j',

age:'22', place:'shivamogga', detail:function()

{

document.write("Name is "+this.fname+" "+this.lname+"<br> age is "+this.age+"<br> place is"+" "+this.place)

}

}

persion.detail()

## Hoisting:

Hoisting is the default behavior of moving all the declarations at the top of the scope before code execution.

**Note**: JavaScript only hoists declarations, not initializations.

JavaScript allocates memory for all variables and functions defined in the program before execution.

console.log(a)//output undefined var a=10;

function declarations are hoisted but function expressions are not hoisted.

sayHello() //output: Hi Good Morning function sayHello(){

console.log("Hi Good Morning")

}

Here the bellow code will result in error

console.log(add(10,20))//error var add=(a,b)=>{

return a+b

}

## Callback Function:

A callback is a function passed as an argument to another function and is invoked after the execution of main function.

### Example:

function getdata(callback){

var name=prompt('enter name') callback(name)

}

function showdata(name){ console.log(name)

}

Here the getdata function will accept a callback function and after that function executed completely the show data function will execute.

### Example 2:

function getAPIData(callback){ console.log("Connecting to API")

setTimeout(function(){ console.log("Accessing data From API") let status="Success"

callback(status)

},4000)

}

function ShowAPIData(status){ console.log("Status From API is "+status)

}

Here getAPIData will execute first and then that will intern return the ShowAPIData method and return the status.

## JavaScript Error handling:

The **try** statement defines a code block to run (to try).

The **catch** statement defines a code block to handle any error.

The **finally** statement defines a code block to run regardless of the result.

The **throw** statement defines a custom error.

**Try** and **catch** will come in hand to hand if any statement that lead to error then place it in try block and the corresponding error will be handled by the catch block.

try {

// Block of code to try

}

catch(error) {

// Block of code to handle errors

}

Example:

try{

console.log("opening file operation") const a=10;

a=20

console.log("closing file")

}

catch(error){ console.log(error.name)

console.log(error.message) console.log(error.stack)//similar to console.log(error)

}

**Finally**: it is a block which will execute irrespective of the error. try {

//Block of code to try

}

catch(err) {

//Block of code to handle errors

}

finally {

//Block of code to be executed regardless of the try / catch result

}

### Example:

try{

console.log("opening file operation") const a=10;

a=20

console.log("closing file")

}

catch(error){ console.log(error.name) console.log(error.message)

console.log(error.stack)//similar to console.log(error)

}

finally{

console.log("closing file operation")

}

## Asynchronous programming

Programming paradigm where control flow of the programme is not determined by the order of the statement in programme but the availability of the resource or data.

## Promise in JavaScript

These are the way to handle the asynchronous operations.

The promise that represent a value that 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 the promise constructor take a function with two parameter resolve and reject.

Here the promise will be either be resolved or it will be rejected.

### Example:

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

{

resolve('Promise Resolved')

}

else{

reject('Promise Rejected')

}

})

## 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.

Eg:

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

resolve('Promise Resolved')

}

else{

reject('Promise Rejected')

}

})

promDemo.then((result)=>console.log(result))

.catch((error)=>{ console.log(error)

})

## Fetch:

Fetch method is used to make a network request or to call a API in Js. Fetch will return a promise in js.

Eg:

***fetch('https://jsonplaceholder.typicode.com/todos/1')***

***.then((result)=>result.json())***

***.then((data)=>console.log(data))***

***.catch((error)=>console.log(error))***

**Callback Hell**

Callback hell is used to describe the nested callback stacked over bellow one another formatting a pyramid structure.

Every callback depends/wait for previous callback.

### Example:

function getUserDetails(userId, callback) { fetch(`/user/${userId}`)

.then(response => response.json())

.then(data => {

callback(null, data);

})

.catch(error => { callback(error, null);

});

}

function getUserPosts(userId, callback) { fetch(`/posts/${userId}`)

.then(response => response.json())

.then(data => { callback(null, data);

})

.catch(error => { callback(error, null);

});

}

function getPostComments(postId, callback) { fetch(`/comments/${postId}`)

.then(response => response.json())

.then(data => { callback(null, data);

})

.catch(error => { callback(error, null);

});

}

getUserDetails(1, (userError, userData) => { if (userError) {

console.error('Error fetching user details:', userError);

} else {

getUserPosts(userData.id, (postsError, userPosts) => { if (postsError) {

console.error('Error fetching user posts:', postsError);

} else { userPosts.forEach(post => {

getPostComments(post.id, (commentsError, comments) => { if (commentsError) {

console.error('Error fetching comments for post:', commentsError);

} else {

console.log('User:', userData); console.log('Post:', post); console.log('Comments:', comments);

}

});

});

}

});

}

});

## 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.

Example:

async function fun1(){

let result=await fetch('https://jsonplaceholder.typicode.com/todos/1')

.then((result)=>result.json())

.then((data)=>console.log(data))

.catch((error)=>console.log(error))

}

fun1()

## 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 a string.

Example:

let obj={

name:`suhas`, age:23,

place: `Bangalore` detail:function(){

console.log(`name is ${this.name} and age is ${this.age} and place is ${this.place}`)

}

}

obj.detail()

## Array Destructuring

It is a feature that allow you to extract the value from the array and assign them to different variable in more concise and reliable manner.

const arr=[10,20,12,13,15] const[a,b,c]=arr console.log(a,b,c)//10,20,12

We can swap the value using Destructuring let a=10;

let b=20

[a,b]=[b,a] Consolelog(a,b)//20,10

**Spread operator in Destructuring.** const arr=[10,20,12,13,15] const[a,b,…c]=arr

console.log(a)//10 console.log(b)//20 console.log(c)//[12,13,15]

## default value

we can provide the default value if value not present. Let a=[10]

const[x,y=20]=a console.log(a,b)//10,20

## object Destructuring

it is a feature that allow you to extract the value from object and assign them from variable.

let obj={ fname:`suhas`, lname:`sj` age:23,

place: `Bangalore` detail:function(){

console.log(`name is ${this.fname} and age is ${this.age} and place is ${this.place}`)

}

}

const[fname,lname]=obj

## Spread operator in Destructuring.

let obj={ fname:`suhas`, lname:`sj` age:23, place:`Bangalore` detail:function(){

console.log(`name is ${this.fname} and age is ${this.age} and place is ${this.place}`)

}

}

const[fname,lname,...detail]=obj

## JavaScript closures

A closure can be defined as a JavaScript feature in which the inner function has access to the outer function variable. In JavaScript, every time a closure is created with the creation of a function.

The closure has three scope chains listed as follows:

* Access to its own scope.
* Access to the variables of the outer function.
* Access to the global variables.

### Example 1:

function outerFunc()

{

var a = 4;

function innerfun()

{

return a;

}

return innerfun;

}

var output=outerFunc() console.log(output())

### Example 2:

function fun(a)

{

function innerfun(b){ return a\*b;

}

return innerfun;

}

var output = fun(4); document.write(output(4));

## Date Object

Date objects are created with the new Date() constructor. let date=new Date()

console.log(date)

Wed Feb 07 2024 17:53:52 GMT+0530 (India Standard Time)

**Methods of date object:**

**getDate()** method is used to get the current date console.log(date.getDate())//print current date

**getMonth()** is used to get the current month. Note: month always start from 0.

console.log(date.getMonth()+1)//current month (start from 0)

**getFullYear()** method is used to print current year. console.log(date.getFullYear())//current year **getHours()** method is used to print the current hour

console.log(date.getHours())//print current hour

**getMinutes()** method is used to print current minutes

console.log(date.getMinutes())//print current minutes

**getSeconds()** method is used to print current seconds. console.log(date.getSeconds())//print current seconds

**Math Object**

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

console.log(Math.PI)//print value of PI console.log(Math.E)//print value of **e** console.log(Math.pow(12,4))**//**print square root of number console.log(Math.random())//print random number from 0 to 1 console.log(Math.floor(2.5))//floor it to nearest integer value console.log(Math.round(2.44))//round to nearest number