**JAVASCRIPT**

1. **Introduction of JavaScript**

* JavaScript is a lightweight, cross-platform, single-threaded, and interpreted compiled programming language.
* It is also known as the scripting language for webpages.
* It is well-known for the development of web pages.
* JavaScript can be used for [Client-side](https://www.geeksforgeeks.org/server-side-client-side-programming/) developments as well as [Server-side](https://www.geeksforgeeks.org/server-side-client-side-programming/) developments.

## **Why JavaScript is known as a lightweight programming language?**

JavaScript is considered lightweight due to the fact that it has low CPU usage, is easy to implement, and has a minimalist syntax. Minimalist syntax as in, has no data types. Everything is treated here as an object.

**Client side:-**

The client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.

**Server side:-**

The server-side extensions allow an application to communicate with a database, and provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.

## How to Link JavaScript File in HTML?

* + Internal JS: We can add JavaScript directly to our HTML file by writing the code inside the <script> tag. The <script> tag can either be placed inside the <head> or the <body> tag according to the requirement.
  + External JS: We can write JavaScript code in another files having an extension.js and then link this file inside the <head> tag of the HTML file in which we want to add this code.

1. **Comments**

* Comments can be used to explain JavaScript code, and to make it more readable.
* Comments can also be used to prevent execution, when testing alternative code.

**Comments are of two types:-**

1. **Single line comment.**

* Single line comments start with //.
* Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

// A paragraph is a series of

// sentences that are organized

// and coherent.

1. **Multi-line comment.**

* Multi-line comments start with /\* and end with \*/.
* Any text between /\* and \*/ will be ignored by JavaScript.

/\* A paragraph is a series of sentences that are organized and coherent, and are all related to a single topic. Almost every piece of writing you do that is longer than a few sentences should be organized into paragraphs. \*/

1. **Console**

Console is an object which provides access to the browser debugging console. We can open a console in a web browser by using Ctrl + Shift + I for windows and Command + Option + K for Mac.

**What is Synchronous and Asynchronous in JavaScript?**

**Synchronous:** Synchronous is a blocking architecture, so the execution of each operation depends on completing the one before it. Each task requires an answer before moving on to the next iteration.

**Asynchronous:** Asynchronous is a non-blocking architecture, so the execution of one task isn't dependent on another. Tasks can run simultaneously.

**Javascript is synchronous or asynchronous language?**

JavaScript is an asynchronous and concurrent programming language that offers a lot of flexibility. It's single-threaded like synchronous but also non-blocking like asynchronous. Although it's synchronous by nature, JavaScript benefits from an asynchronous process.

**Is JavaScript by default synchronous or asynchronous?**

JavaScript is synchronous by default: every line of code is executed one after the other, and each task must wait for the previous one to be completed before moving to the next.

**What is single Thread and Multi-Thread in Javascript?**

**Single Thread**: A single-threaded language is one that can execute only one task at a time. The program will execute the tasks in sequence, and each task must complete before the next task starts.

**Multi Thread:** Multi-threading is the ability of any program to execute multiple threads simultaneously.

**Is JavaScript single-threaded or multi-threaded?**

JavaScript is a popular single-threaded programming language commonly used for web development. In web browsers, JavaScript runs in the main thread and handles tasks like user interactions, DOM manipulation, and HTTP requests.

1. **JavaScript Variables**

Variables are containers for storing information. Creating a variable in JavaScript is called "declaring". When we provide a value to a variable then it is called “assign”.

**Types of Variable**

* Var
* let
* const
* Automatically

**Automatically**:-

x = 5;

y = 6;

z = x + y;

console.log(z);

x = 5;

x = 6;

console.log(x)

**Var**:-

var x = 5;

var y = 6;

var z = x + y;

console.log(z)

var x = 5;

var x = 6;

console.log(x)

Var is a global scope variable so we can re-declare the variable again and we can re-assign the value.

**let** :-

Let x = 5;

Let y = 6;

Let z = x + y;

console.log(z)

let x = "abc";  
let x = 0;

Console.log(x)

Error:- SyntaxError: Identifier 'x' has already been declared

let x = "abc";  
 x = 0;

Console.log(x)

Let is a block scope variable so we can not re-declare the variable again but we can re-assign the value.

**const** :-

Const x = 5;

Const y = 6;

Const z = x + y;

console.log(z)

const x = 5;

const x = 6;

console.log(x)

Error:- SyntaxError: Identifier 'x' has already been declared

const x = 5;

x = 6;

console.log(x)

Error:- TypeError: Assignment to constant variable.

Use const when you declare:

* A new Array
* A new Object
* A new Function

Const is also a block scope variable but we cannot re-declare the variable again also we cannot re-assign the value.

**Data Types**

Data types in JavaScript define the data type that a variable can store.

### JavaScript has 2 Datatypes

**Primitive:-**

1. String

let str = "Hello There";

2. Number

let num = 2; // Integer

3. Bigint

let bigBin = BigInt("0b1010101001010101001111111111111111");

4. Boolean

let a = true; // yes  
 let b = false; // no

5. Undefined

let x;  
console.log(x); // undefined

6. Null

let age = null;

7. Symbol

let symbol1 = Symbol("Geeks")

**Non-Primitive:-**  
1. Object

let person = {

Name:”abc”

}

1. Array

Let arr = [1,2,3,4,5]

1. **Operators**

Javascript operators are used to perform different types of mathematical and logical computations.

The **Assignment Operator** **=** assigns values

The **Addition +** adds values

The **Multiplication \*** multiplies values

The **Comparison Operator** **>** compares values

1. **Arithmetic Operators**

**Arithmetic Operators** are used to perform arithmetic on numbers:

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| \*\* | Exponentiation ([ES6](https://www.w3schools.com/js/js_2016.asp)) |
| / | Division |
| % | Modulus (Division Remainder) |
| ++ | Increment |
| -- | Decrement |

## Assignment Operators

Assignment operators assign values to JavaScript variables.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |
| \*\*= | x \*\*= y | x = x \*\* y |

**var x = 10;**

**x += 5;**

**console.log(x)**

**Output: 15**

## Comparison Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | equal to |
| === | equal value and equal type |
| != | not equal |
| !== | not equal value or not equal type |
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |
| ? | ternary operator |

1. **IF-Else Condition**

# if, else, and else if

* Use if to specify a block of code to be executed, if a specified condition is true
* Use else to specify a block of code to be executed, if the same condition is false
* Use else if to specify a new condition to test, if the first condition is false

**Let hour = 20;**

if (hour < 18) {  
   console.log("Good day");  
}

Let hour = 17;

if (hour < 18) {  
  console.log( "Good day" );  
} else {  
 console.log("Good evening");  
}

Let hour = 21

if (hour > 17) {  
  console.log( "Good day" );  
} else if(hour <= 20){

  console.log( "Good noon" );}

else {  
  console.log("Good evening");  
}

1. **Array**

An array is a type of global object that is used to store data. Arrays consist of an ordered collection or list containing zero or more data types

let a = [1,2,3,"a",4,"b",true]

console.log(a)

We can stored value like this

let car1 = "yundai";  
let car2 = "maruti";  
let car3 = "honda";

But what if we have more than 3 or 30 values then we use array:-

Let car = ["yundai","maruti","honda",”BMW”,”AUDI”];

1. **Object**

An object is a collection of properties, and a property is an association between a key and a value.

let student= {

firstName:”abc”,

lastName:”xyz”,

age:20,

email:”abc@gmail.com”

};

1. **Array Of Object**

The Array of object enables storing a collection of multiple items under a single variable.

let arr = [

{name:"abc", email:"abc@gmail.com",pass:"unique" },

{name:"xyz", email:"xyz@gmail.com",pass:"123456789"}

]

1. **Array Methods**

**Length():-**

const number = [1,2,3,4,5];

let num = number.length

console.log(num);

Output:- 5

**toString():-**

**Covert array in string**

const fruits = ["Banana", "Orange", "Apple", "Mango"];

const fruit = fruits.toString();

console.log(fruit);

Output:- Banana,Orange,Apple,Mango

**Array at():-**

const fruits = ["Banana", "Orange", "Apple", "Mango"];

let fruit = fruits.at(2);

Output:-Apple

**Join():-**

const fruits = ["Banana", "Orange", "Apple", "Mango"];

let fruit = fruits.join(" - ");

console.log(fruit);

Output:-

## push():-

const developer= ["ravi", "ajay", "shiv", "raj"];

developer.push("ayush");

console.log(developer);

Output:-

**pop():-**

const developer= ["ravi", "ajay", "shiv", "raj"];

developer.pop();

console.log(developer);

Output:-

## unshift():-

const developer= ["ravi", "ajay", "shiv", "raj"];

developer.unshift("komal");

console.log(developer);

Output:-

**shift():-**

const developer= ["ravi", "ajay", "shiv", "raj"];

developer.shift();

console.log(developer);

Output:-

## Change Array Elements:-

const developer= ["ravi", "ajay", "shiv", "raj"];

developer[0] = "shailesh";

Console.log(developer)

Output:-

**Delete():-**

const fruits = ["Banana", "Orange", "Apple", "Mango"];

delete fruits[0];

console.log(fruits)

Output:-  [empty, 'Orange', 'Apple', 'Mango']

Using delete() leaves undefined holes in the array.

## concat():-

const fruit=['apple','grapes','pineapple']

const vegetable=['potato','tomato']

const combo = vegetable.concat(fruit);

console.log(combo);

**flat():-**

The flat() method creates a new array with sub-array elements by concating

const myArr = [[1,2],[3,4],[5,6,[10,20]];

const newArr = myArr.flat(2);

console.log(newArr);

**slice()**

The slice() method creates a new array.

The slice() method does not remove any elements from the existing array.

const alphabet = ["a", "b", "c", "d", "e"];

const newAlpha = alphabet.slice(2);

console.log(newAlpha);

const alphabet = ["a", "b", "c", "d", "e"];

const newAlpha = alphabet.slice(2, 2);

console.log(newAlpha)

First parameter:- Defines the starting value, means the beginning of the array.

Second parameter:- Defines the index, till that index we will get the output but will not include the selected index.

**splice():-**

const alphabet = ["a", "b", "c", "d", "e","f","g"];

alphabet.splice(3,1);

console.log(alphabet);

The first parameter () specifies the index at which to start changing the array.

The second parameter () defines **how many** elements should be **removed**.

## toSpliced()

**sort():-**

const fruits = ["z", "a", "m", "c"];

fruits.sort();

console.log(fruits);

let array = [4, 2, 1, 7, 8, 0];

array.sort((a, b) => a - b);

console.log(array); //ascending

let array = [4, 2, 1, 7, 8, 0];

array.sort((a, b) => b - a);

console.log(array); //descending

**toSorted()**

**reverse():-**

const fruits = ["z", "a", "m", "c"];

fruits.reverse();

console.log(fruits);

**toReversed()**

**indexOf():-**

const fruits = ["Appl", "Orange", "Apple", "Mango"];

let position = fruits.indexOf("Apple");

console.log(position);

Output:- 2

**lastIndexOf():-**

const fruits = ["Apple", "Orange", "Apple", "Mango"];

let position = fruits.lastIndexOf("Apple");

console.log(position)

Output:- 2

1. **Map**

Map creates a new array by calling a function on every element of the original array and storing the results in a new array. map() returns the new array.

1). let a = [1,2,3]

let b = a.map((value,index,array)=>{

    return value \* value

     console.log(value)

});

console.log(b);

Output:- [1,4,9]

2). let a = [1,2,3]

let b = a.map((value,index,array)=>{

     return value \* index

     console.log(value)

})

console.log(b);

Output:- [0,2,6];

3). let a = [1,2,3]

let b = a.map((value,index,array)=>{

    return value \* array

     console.log(value)

})

console.log(b);

Output:- [ NaN, NaN, NaN ]

4). let a = [1,2,3]

let b = a.map((value,index,array)=>{

     return value + value

     console.log(value)

})

console.log(b);

Output:- [ 2, 4, 6 ]

5). let a = [1,2,3]

let b = a.map((value,index,array)=>{

     return value + index

     console.log(value)

})

console.log(b);

Output:- [ 1, 3, 5 ]

6). let a = [1,2,3]

let b = a.map((value,index,array)=>{

     return value + array

     console.log(value)

})

console.log(b);

Output:- [ '11,2,3', '21,2,3', '31,2,3' ]

7). let a = [1,2,3]

let b = a.map((value,index,array)=>{

     return value \*\* value

    console.log(value)

})

console.log(b);

Output:- [ 1, 4, 27 ]

8). let a = [1,2,3]

let b = a.map((value,index,array)=>{

     return value \*\* index

})

console.log(b);

Output:- [ 1, 2, 9 ]

9). let a = [1,2,3]

let b = a.map((value,index,array)=>{

     return value \*\* array

})

console.log(b);

Output:- [ NaN, NaN, NaN ]

2. **Filter**

The filter() method creates a new array filled with elements that pass a test provided by a function. The filter() method does not execute the function for empty elements.

const ages = [32, 33, 12, 40];

let age = ages.filter((value)=>{

   return value < 15

})

console.log(age)

Output:- 12

3. **Reduce**

Reduce() in JavaScript is a predefined method used to reduce an array to a single value by passing a callback function on each element of the array. It accepts a function executed on all the items of the specified array in the left-to-right sequence.

let a = [1,2,3,9,4]

let b = a.reduce((value,index)=>{

return value + index;

})

console.log(b);

Output:- 19

let a = [1,2,3,9,4]

let b = a.reduce((value,index,array)=>{

return value + value

})

console.log(b);

Output:- 16

let a = [1,3,0]

const addition = ((value, index)=>{

return value + value

})

let b = a.reduce(addition );

console.log(b);

**Events**:

JavaScript events are actions or occurrences (like clicks, key presses, or form submissions) that happen in the browser, allowing scripts to execute specific responses. They enable interactive and dynamic web page behaviors by attaching event listeners to elements.

· onclick: Fired when an element is clicked.

· ondblclick: Fired when an element is double-clicked.

· onsubmit: Fired when a form is submitted.

· onchange: Fired when the value of an element changes (e.g., input, select).

· onmouseout: Fired when the mouse pointer is moved out of an element.

· onmouseover: Fired when a form is submitted.

**Functions**

**JavaScript functions** are used to perform operations. We can call JavaScript function many times to reuse the code. A function is defined as a relation between a set of inputs having one output each.

#### Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

1. **Code re-usability**: We can call a function several times so it save coding.
2. **Less coding**: It makes our program compact. We don’t need to write many lines of code each time to perform a common task.

### [Different types of functions in JavaScript?](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[There are three main types of functions in JavaScript:](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

* [named functions](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)
* [anonymous functions](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)
* [arrow functions](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[Syntax:-](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

[function studentData(){](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[}](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[A JavaScript function is defined with the function keyword, followed by a](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**[name](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**[, followed by parentheses](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**[()](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**[.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[The code inside the function will execute when "something"](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**[invokes](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**[(calls) the function:](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

* [When an event occurs (when a user clicks a button).](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)
* [When it is invoked (called) from JavaScript code.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)
* [Automatically (self invoked).](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[We can write the regular function in two ways, i.e.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**[Function declaration](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**[and](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**[Function expression.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[//function declaration](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[console.log(add(2,3));](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[function add(a,b){](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[return a+b;](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[};](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[console.log(add(2,3));](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[//function expression](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[console.log(sum(2,3));](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[let sum= function(a,b){](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[return a+b;](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[};](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[console.log(sum(2,3));](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[Function expression can be anonymous, while function declaration must have a name.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[We can not access expression function before initialization.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[Error:- Uncaught ReferenceError: Cannot access 'sum' before initialization](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[Anonymous Function:-](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

[An anonymous function is that type of function that has no name or we can say which is without any name](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[(function () {](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[console.log("Hello Guys");](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[})();](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[Arrow Function:-](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

[A new and shorter way of declaring an anonymous function, which is known as](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)[[Arrow Functions.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)](https://www.geeksforgeeks.org/arrow-functions-in-javascript/)[In an Arrow function, everything remains the same, except here we don’t need the](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)*[function](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)*[keyword.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[let](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)** [hello= () =>{](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[console.log("Hello");](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[};](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[hello();](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[If we have only a single statement in the function body, we can even remove the curly braces.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[let hello= () => console.log("Hello");](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[hello();](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[let hello = (()=>{](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[})](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[In Javascript we have two types of storage:](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

[1. LocalStorage](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[2. SessionStorage](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[LocalStorage:](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

[localStorage in JavaScript allows web applications to store data locally within the user's browser – with no expiration date. The data isn't deleted when the browser is closed, and is available when the browser is opened again.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[What is the use of local storage?](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

[localStorage can store form data that won't be lost if the user closes the browser. If you have an application that requires you to log in, localStorage can be used to keep your session data. You can remain logged in even after closing and reopening the browser.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[What are 3 advantages of local storage?](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

[Local storage offers the advantages of control, privacy, performance, offline access, and dependability. The user has complete control over the data and storage infrastructure, which might be necessary for security and regulatory reasons.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[What type is local storage?](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

[The localStorage is a browser API that is used to store key-value pairs in the browser. The localStorage API is a read-only memory in which the data can be read but cannot be modified.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[Two methods to use Localstorage:](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[1. setItem](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[Syntax](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**[:- localStorage.setItem("key-name",variable-name);](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[localStorage.setItem("key-name",JSON.stringify(variable-name));](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[2. getItem](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

**[Syntax](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**[:- localStorage.getItem("key-name");](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[JSON.parse(localStorage.getItem("key-name"));](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[JSON:](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

[JSON stands for JavaScript Object Notation.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[JSON is a text format for storing and transporting data.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[Two Methods:](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

[JSON.parse()](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[JavaScript has a built in function for converting JSON strings into JavaScript objects:](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[JSON.stringify()](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[JavaScript also has a built in function for converting an object into a JSON string:](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[Delete item from Localstorage:](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

[localStorage.clear();](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[Delete Perticular key from localstorage:](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

[localStorage.removeItem("key-name");](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

**[Hoisting:](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)**

[What is Hoisting ?](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[Hoisting is Javascript's default behaviour of moving declarations to the top.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

[Declarations can be both variable and function.](https://reintech.io/blog/types-of-functions-in-javascript" \l ":~:text=In this example, the function,anonymous functions, and arrow functions.)

1. **Currying**

* A process of converting a function that takes multiple arguments into a sequence of functions is called currying function.
* The curried function returns a new function that expects the next argument in sequence.

**Why we use currying function ?**

 Currying is a technique in JavaScript that allows you to transform functions with multiple arguments into a sequence of functions, each taking one argument at a time. It promotes code re-usability and flexibility.

//Normal Function

function addition(a,b,c){

return a+b+c;

}

console.log(addition(1,2,3));

function add(a){

return function(b){

return function(c){

return a+b+c;

}

}

}

let sum = add(2);

let sum1= sum(3);

let sum2= sum1(4);

console.log(sum2);

// With currying function

console.log(add(2)(3)(4));

//infinite Currying Function

function adds(a){

return function(b){

if(b) return adds(a+b);

return a;

}

}

console.log(adds(10)(20)(30)(10)());

**Destructuring of Array:**

 JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables. We can extract a value from an array and put them into other variables. Array destructuring javascript is also used to assign and declare a variable.

**Destructuring of Object:**

a useful JavaScript feature to extract properties from objects and bind them to variables. Even better, object destructuring can extract multiple properties in a single statement, can access properties from nested objects, and can set a default value if the property doesn't exist.

**Template Literals:**

A template literal is a feature in JavaScript that allows you to embed expressions in strings using backticks (`). It is also known as a template string. It helps to concatenate strings with variables or expressions without the need for string concatenation.

**Rest Operator:**

The rest operator in Javascript, represented by three dots (...), enables you to collect a series of items. and gathers all elements that match a certain condition into an array.

1. **Loops**

Loops can execute a block of code a number of times.if you want to run the same code over and over again, each time with a different value.

## Different Kinds of Loops

* For loop

Loops through a block of code a number of times

for (*expression 1*;*expression 2*;*expression 3*) {  
  // *code block to be executed*  
}

for (let i = 0; i < 5; i++) {

console.log(i)

}

Expression 1 sets a variable before the loop starts (let i = 0).

Expression 2 defines the condition for the loop to run (i must be less than 5).

Expression 3 increases a value (i++) each time the code block in the loop has been executed.

* While loop

The while loop loops through a block of code as long as a specified condition is true.

Let i = 1;

While(I<=5){

console.log(i);// if we don’t give increment then it will work infinite times

i++;

};

* Do While loop

This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax:- do {  
*(statement)* } while (condition);

let i = 0;

do {

console.log(i);

i++;

}

while (i < 10);

let password;

do{

password = prompt("Enter the password");

} while( password !=="anurag" );

**Without Callback**

function sayHello(){

console.log("Hello")

};

function add(num1,num2){

console.log(num1 + num2);

};

add(2,4,sayHello());

**With Callback**

**1).** function sayHello(){

console.log("Hello")

};

function add(num1,num2,callback){

console.log(num1 + num2);

callback();

}

add(2,4,sayHello);

2). function sayHello(){

console.log("Hello")

};

function sayHi(){

console.log("Hi")

}

function add(num1,num2,callback){

console.log(num1 + num2);

callback();

}

add(2,4,sayHello);

add(100,4,sayHi);

**We can also pass anonyomous function**

function sayHello(){

console.log("Hello")

};

function sayHi(){

console.log("Hi")

};

function add(num1,num2,callback){

console.log(num1 + num2);

callback();

}

add(2,4,sayHello);

add(100,4,sayHi);

add(2000,4,function(){

console.log("Session End")

});

1. **Callback Hell**

Callback hell in Javascript occurs when multiple callbacks are nested inside each other, leads to create a complex and hard-to-manage code. Which makes it difficult to follow the flow of the program.

setTimeout(()=>{

console.log("First task is done.");

setTimeout(()=>{

console.log("Second task is done.");

setTimeout(()=>{

console.log("Third task is done.");

setTimeout(()=>{

console.log("Fourth task is done.");

setTimeout(()=>{

console.log("Fifth task is done.");

setTimeout(()=>{

console.log("sixth task is done.");

},3000);

},3000);

},3000);

},3000);

},3000);

},3000);

**Pyramid Of Doom**

Callback Hell is also known as the “Pyramid of Doom”. Is a situation where nested callbacks lead to deeply indented and hard-to-read code. It can make your code look like a pyramid due to its visual structure.

setTimeout(() => {

  console.log("First task is done.");

  setTimeout(() => {

    console.log("Second task is done.");

    setTimeout(() => {

      console.log("Third task is done.");

      setTimeout(() => {

        console.log("Fourth task is done.");

        setTimeout(() => {

          console.log("Fifth task is done.");

          setTimeout(() => {

            console.log("sixth task is done.");

          }, 3000);

        }, 3000);

      }, 3000);

    }, 3000);

  }, 3000);

}, 3000);

1. **Promises**

let data = new Promise((resolve,reject)=>{

setTimeout(()=>{

resolve("Your promise is resolved")

},2000)

})

data.then((item)=>{

console.log(item)

}).catch((err)=>{

console.log("Caught error",err)

});

## Promise.all()

Promise.all method in JavaScript is used to handle multiple promises concurrently and wait for all of them to resolve.

const promise1 = new Promise((resolve) =>

setTimeout(() => resolve("Promise 1 resolved"), 1000)

);

const promise2 = new Promise((resolve) =>

setTimeout(() => resolve("Promise 2 resolved"), 500)

);

const promise3 = new Promise((resolve) =>

setTimeout(() => resolve("Promise 3 resolved"), 800)

);

const promisesArray = [promise1, promise2, promise3];

Promise.all(promisesArray)

.then((results) => {

console.log("All promises resolved:", results);

})

.catch((error) => {

console.error("At least one promise rejected:", error);

});

Output:- All promises resolved: [ 'Promise 1 resolved', 'Promise 2 resolved', 'Promise 3 resolved' ]

**If any of the promise rejects, it won't wait for others to resolve and will return with reason of the promise that is rejected.**

const promise1 = new Promise((resolve) =>

setTimeout(() => resolve("Promise 1 resolved"), 1000)

);

const promise2 = new Promise((resolve) =>

setTimeout(() => resolve("Promise 2 resolved"), 500)

);

const promise3 = new Promise((\_, reject) =>

setTimeout(() => reject("Promise 3 rejected"), 800)

);

const promisesArray = [promise1, promise2, promise3];

Promise.all(promisesArray)

.then((results) => {

console.log("All promises resolved:", results);

})

.catch((error) => {

console.error("At least one promise rejected:", error);

});

Output:- At least one promise rejected: Promise 1 rejected

## Promise.allSettled()

The Promise.allSettled method is used to handle multiple promises concurrently, just like Promise.all, but it waits for all the promises to settle (either resolve or reject) before proceeding. It returns an array of objects.

Promise.allSettled() always returns array of objects with status key which denotes fulfilled or rejected. If a promise is fulfilled then you can get response with value key and if the promise is rejected then you can find the reason in reason key.

const promise1 = new Promise((resolve) =>

setTimeout(() => resolve("Promise 1 resolved"), 1000)

);

const promise2 = new Promise((\_, reject) =>

setTimeout(() => reject("Promise 2 resolved"), 500)

);

const promise3 = new Promise((resolve) =>

setTimeout(() => resolve("Promise 3 resolved"), 800)

);

const promisesArray = [promise1, promise2, promise3];

Promise.allSettled(promisesArray).then((results) => {

console.log("All promises settled:", results);

});

Output:- [

{ status: 'fulfilled', value: 'Promise 1 resolved' },

{ status: 'rejected', reason: 'Promise 2 resolved' },

{ status: 'fulfilled', value: 'Promise 3 resolved' }

]

## Promise.race()

## Promise.race() returns first promise with shortest delay whether it is resolved or rejected.

const promise1 = new Promise((resolve) =>

setTimeout(() => resolve("Promise 1 resolved"), 1000)

);

const promise2 = new Promise((\_,reject) =>

setTimeout(() => reject("Promise 2 rejected"), 500)

);

const promise3 = new Promise((resolve) =>

setTimeout(() => resolve("Promise 3 resolved"), 800)

);

const promisesArray = [promise1, promise2, promise3];

Promise.race(promisesArray)

.then((results) => {

console.log("First promise resolved:", results);

})

.catch((error) => {

console.error("At least one promise rejected:", error);

});

Output:- At least one promise rejected: Promise 2 rejected

## Promise.any()

It will return with **first resolved promise**.

const promise1 = new Promise((resolve) =>

setTimeout(() => resolve("Promise 1 resolved"), 1000)

);

const promise2 = new Promise((\_,reject) =>

setTimeout(() => reject("Promise 2 rejected"), 500)

);

const promise3 = new Promise((resolve) =>

setTimeout(() => resolve("Promise 3 resolved"), 800)

);

const promisesArray = [promise1, promise2, promise3];

Promise.any(promisesArray)

.then((results) => {

console.log("First promise resolved:", results);

})

.catch((error) => {

console.error("At least one promise rejected:", error);

});

Output:- First promise resolved: Promise 3 resolved

1. **Async and Await**

The async keyword transforms a regular JavaScript function into an asynchronous function, causing it to return a Promise. The await keyword is used inside an async function to pause its execution and wait for a Promise to resolve before continuing.

**Can we use await without async?**

The await operator is used to wait for a Promise and get its fulfillment value. It can only be used inside an async function or at the top level of a module.

**Why use await in JavaScript?**

It tells JavaScript to wait for an asynchronous action to finish before continuing the function. It's like a 'pause until done' keyword.

**Call Apply Bind**

**Without using Method:-**

let obj={

name:"abc"

};

function myFun(obj,fullName){

return "hello" + " "+ obj.name +" "+ fullName

}

console.log(myFun(obj,"xyz"));

**//Call**

The call() method calls the function directly and sets this to the first argument passed to the call method and if any other sequences of arguments preceding the first argument are passed to the call method then they are passed as an argument to the function.

let obj={

name:"abc"

};

function inherit(lastName){

return 'Hello' + " " + this.name + " " + lastName

};

const data = inherit.call(obj,"xyz");

console.log(data);

Output:- Hello abc xyz

**//Apply**

The apply method is used to call a function in a different object with the given this value, and the arguments are passed in the form of an array.

let obj={

name:"abc"

};

function inherit(lastName,status){

return 'Hello' + " " + this.name + " " + lastName + " I am "+ status

};

const data = inherit.apply(obj,["xyz", "good"]);

console.log(data);

**What is the difference between call () and apply ()?**

The call() method takes arguments separately. The apply() method takes arguments as an array. The apply() method is very handy if you want to use an array instead of an argument list.

**//Bind**

Creates a new function that, when called, calls this function with its this keyword set to the provided value.

let obj={

name:"abc"

};

function inherit(lastName,status){

return 'Hello' + " " + this.name + " " + lastName + " I am "+ status

};

const data = inherit.bind(obj,"xyz", "good");

console.log(data());