What is JavaScript?

JavaScript is a **programming language used for creating dynamic content on websites**. It is a **lightweight**, **cross-platform** and **single- threaded** programming language. JavaScript is an **interpreted** language that executes code line by line providing more flexibility. It is a commonly used programming language to **create dynamic and interactive elements in web applications**. It is easy to learn. Itis a **weakly typed language (dynamically typed)**. JavaScript can be used for **Client-side** developments as well as **Server-side** developments. JavaScript is both an imperative and declarative type of language. JavaScript contains a standard library of objects, like **Array**, **Date**, and **Math**, and a core set of language elements like **operators**, **control structures**, and **statements**.

• Client-side: It supplies objects to control a browser and its Document Object Model

(DOM). Like if 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. Useful libraries for the client side are AngularJS, ReactJS, VueJS, and so many others.

• Server-side: It supplies objects relevant to running JavaScript on a server. For if 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. The useful framework

which is the most famous these days is node.js.

• Server-side: It supplies objects relevant to

running JavaScript on a server. For if 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. The useful framework which is the most famous these days is node.js.

- Imperative language – In this type of language we are mostly concerned about how it is to be done. It simply controls the flow of computation. The procedural programming approach, object, oriented approach comes under this as async await we are thinking about what is to be done further after the async call.

- Declarative programming – In this type of language we are concerned about how it is to be done, basically here logical computation requires. Her main goal is to describe the desired result

without direct dictation on how to get it as the arrow function does.

History of JavaScript:

• It was created in 1995 by Brendan Eich while he was an engineer at Netscape. It was originally going to be named LiveScript but was renamed. Unlike most programming languages, JavaScript language has no concept of input or output. It is designed to run as a scripting language in a host environment, and it is up to the host environment to provide mechanisms for

communicating with the outside world. The most

common host environment is the browser.

Features of JavaScript:

According to a recent survey conducted by **Stack Overflow**, JavaScript

is the most popular language on earth.

With advances in browser technology and JavaScript having moved into the server with Node.js and other frameworks, JavaScript is capable of so much more. Here are a few things that we can do with JavaScript:

- JavaScript was created in the first place for DOM manipulation. Earlier websites were mostly static, after JS was created dynamic Web sites were made.

- Functions in JS are objects. They may have properties and methods just like other objects. They can be passed as arguments in other functions.

- Can handle date and time.

- Performs Form Validation although the

forms are created using HTML.

- No compiler is needed.

Applications of JavaScript:

• **Web Development:** Adding interactivity and behaviour to static sites JavaScript was invented to do this in 1995. By using AngularJS that can be achieved so easily.

• **Web Applications:** With technology, browsers have improved to the extent that a language was required to create robust web applications. When we explore a map in Google Maps then we only need to click and drag the mouse. All detailed view is just a click away, and this is possible only because of JavaScript. It uses Application Programming Interfaces(APIs) that provide extra power to the code. The Electron and React are helpful in this department.

- **Server Applications: With the help of Node.js, JavaScript made its way from client to server and Node.js is the most powerful on the server side.**

- Games: Not only in websites, but JavaScript also helps in creating games for leisure. The combination of JavaScript and HTML 5 makes JavaScript popular in game development as well. It provides the EaseJS library which provides solutions for working with rich graphics.

- Smartwatches: JavaScript is being used in all

possible devices and applications. It provides a

library PebbleJS which is used in smartwatch

applications. This framework works for applications that require the Internet for their functioning.

- Art: Artists and designers can create whatever they want using JavaScript to draw on HTML 5 canvas, and make the sound more effective also can be used p5.js library.

- Machine Learning: This JavaScript ml5.js library can be used in web development by using machine learning.

- Mobile Applications: JavaScript can also be used to build an application for non-web contexts. The features and uses of JavaScript make it a powerful tool for creating mobile applications. This is a Framework for building web

and mobile apps using JavaScript. Using React Native, we can build mobile applications for different operating systems. We do not require to write code for different systems. Write once use it anywhere!

Limitations of JavaScript:

• Weak error handling and type checking facilities: It is a weakly typed language as there is no need to specify the data type of the variable. So wrong type checking is not performed by compile.

- Security risks: JavaScript can be used to fetch data using AJAX or by manipulating tags that load data such as <img>, <object>, <script>. These attacks are called cross-site script attacks. They inject JS that is not part of the site into the visitor's browser thus fetching the details.

- Performance: JavaScript does not provide the same level of performance as offered by many traditional languages as a complex program written in JavaScript would be comparatively slow. But as JavaScript is used to perform simple tasks in a browser, so performance is not considered a big restriction in its use.

- Complexity: To master a scripting language, programmers must have a thorough knowledge of all the programming concepts, core language objects, and client and server-side objects otherwise it would be difficult for them to write advanced scripts using JavaScript.

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. It is very easy to learn because of its syntax similar to C++ and Java.

A lightweight language does not consume much of your CPU’s resources. It doesn’t put excess strain on your CPU or RAM. JavaScript runs in the browser even though it has complex paradigms and logic which means it uses fewer resources than other languages. For example, NodeJs, a variation of JavaScript not only performs faster computations but also uses fewer resources than its counterparts such as Dart or Java.

Additionally, when compared with other

programming languages, it has fewer in-built libraries or frameworks, contributing as another reason for it being lightweight. However, this brings a drawback in that we need to incorporate external libraries and frameworks.

Console:

A **web console** is a tool that is mainly used to log information associated with a web page like, network requests, JavaScript, security errors, warnings, CSS, etc. It enables us to interact with a web page by executing JavaScript expressions in the contents of the page.

Console object: In JavaScript, a 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.

JavaScript Console Methods: The console object provides us with several

different methods.

- JS console.log() Method
- JS console.error() Method
- JS console.warn() Method
- JS console.clear() Method
- JS console.time() and console.timeEnd() Method
- JS console.table() Method
- JS console.count() Method
- JS console.group() and

console.groupEnd() Method

- JS Custom console logs
  JavaScript console.log() Method: It is used to log(print) the output to the console. We can put any type inside the log(), be it a string, array, object, boolean etc.
  Example:
  // console.log() method
  console.log('abc');
  console.log(1);
  console.log(**true**);
  console.log(**null**);
  console.log(undefined);
  console.log([1, 2, 3, 4]); // array inside log console.log({a:1, b:2, c:3}); // object inside **JavaScript console.error() Method:** This method is used to log an error message to the console. Useful in testing of code. By default the error message will be highlighted with red colour.
  Example:
  // console.error() method

- console.error('This is a simple error'); • **JavaScript console.warn() Method:** Used to log warning message to the console. By default, the warning message will be highlighted with yellow colour.
  Example:
  // console.warn() method

- console.warn('This is a warning.'); •

// console.warn() method
• console.warn('This is a warning.'); •

JavaScript console.clear() Method:
Used to clear the console. The console will be cleared, in case of Chrome a simple overlayed text will be printed like : ‘Console was cleared’ while in Firefox no message is returned.

Example:

```
// console.clear() method
console.clear();
```

JavaScript console.time() and console.timeEnd() Method:

Whenever we want to know the amount of time spend by a block or a function, we can make use of the time() and timeEnd() methods provided by the JavaScript console object. They take a label which must be same, and the code inside can be

anything(function, object, simple console).

JavaScript Syntax:

JavaScript syntax refers to the rules and conventions governing the structure and arrangement of code in the JavaScript programming language. It includes statements, expressions, variables, functions, operators, and control flow constructs.

Syntax

```
console.log("Basic Print method in JavaScript");
```

JavaScript syntax refers to the set of rules that

determines how JavaScript programs are constructed:

```
// Variable declaration
let c, d, e;
// Assign value to the variable c = 5;
// Compute value of variables d = c;
e = c / d;
```

JavaScript Values:

There are two types of values defined in JavaScript **Syntax**:

- **Fixed Values: These are known as the literals.**

- Variable values: These are called variables

JavaScript Literals:

•

Syntax Rules for the JavaScript fixed values are: JavaScript Numbers can be written with or without decimals. Javascript Strings are text that can be written in single or double quotes.
let num1 = 50 let num2 = 50.05 let str1 = "hello" let str2 = "
console.log(num1)

•

console.log(num2)

.

•

•

Local variables: Declare a variable inside of a block or function.

Global variables: Declare a variable outside function or with a window object. Example: This example shows the use of Javascript variables.

// Declare a variable and initialise

•

// Global variable declaration

•

let Name = "Apple";

// Function definition

function MyFunction() { •

// Local variable declaration let num = 45; *// Display the value of Global variable*

•

```
 console.log(Name);
 // Display the value of local
• variable
• •
```

Output: Apple 45

JavaScript Operators:

JavaScript operators are symbols that are used to compute the value or in other words, we can perform operations on operands. Arithmetic operators (+, -, *, /) are used to compute the value, and Assignment operators (=, +=, %=) are used to assign the values to variables.

Example: This example shows the use of javascript operators. *// Variable Declarations*

```
let x, y, sum;
// Assign value to the variables
x = 3; y = 23;
```

// Use arithmetic operator to // add two numbers

sum = x + y; console.log(sum); **Output**

console.log(num); }
// Function call MyFunction();

JavaScript Expressions:

Javascript Expression is the combination of values, operators, and variables. It is used to compute the values.

Example: This example shows a JavaScript expression.

// Variable Declarations

let x, num, sum;
// Assign value to the variables

x = 20; y = 30

*// **Expression to divide a number*** num = x / 2;

// Expression to add two numbers

Apple 45

sum = x + y;
console.log(num + "\n" + sum); **Output**

10 50

JavaScript Keywords:

The keywords are the reserved words that have special meanings in JavaScript.

(single-line comment) or between /* and */ (multi-line comment) is treated as a comment and

ignored by the JavaScript compiler. **Example:** This example shows the use of javascript comments.

// Variable Declarations

let x, num, sum;
// Assign value to the variables

x = 20; y = 30

/ Expression to add two numbers */*

sum = x + y; console.log(sum); **Output**

50

JavaScript Data Types:

JavaScript is a **dynamically typed** (also called loosely

typed) scripting language. In JavaScript, variables can receive different data types over time.

The latest ECMAScript standard defines eight data types Out of which seven data types are **Primitive(predefined)** and one **complex or Non- Primitive**.

Primitive Data Types

The predefined data types provided by JavaScript language are known as primitive data types. Primitive data types are also known as in-built data types.

- **Number: JavaScript numbers are always stored in double- precision 64-bit binary format IEEE 754. Unlike other programming languages, you don't need int, float, etc to declare different numeric values.**

- **String: JavaScript Strings are similar to sentences. They are made up of a list of characters, which is essentially just an "array of characters, like "Hello"**

• Boolean: Represent a logical entity and can

have two values: true or false.

- **Null: This type has only one value that is null.**

- **Undefined: A variable that has not been assigned a value is undefined.**

- **Symbol: Symbols return unique identifiers that can be used to add unique property keys to an object that won’t collide with keys of any other code that might add to the object.**

- **BigInt: BigInt is a built-in object in**

JavaScript that provides a way to represent whole numbers larger than 253-1.
JavaScript Primitive Data Types Examples: Examples: Number: The number type in JavaScript contains both integer and floating-point numbers. Besides these numbers, we also have some 'special- numbers' in javascript that are: 'Infinity', '- Infinity', and 'NaN'. Infinity basically represents the mathematical '?'. The 'NaN' denotes a computational error.
let num = 2; // Integer

- **let num2 = 1.3; // Floating point number**

- **let num3 = Infinity; // Infinity**

- **let num4 = 'something here too'/2; //**

NaN •
String:
A String in javascript is basically a series of characters that are surrounded by quotes. There are three types of quotes in Javascript, which are:
let str = "Hello There";

- **let str2 = 'Single quotes works fine';**

- **let phrase = `can embed ${str}`;**

The special null value does not belong to any of the default data types. It forms a separate type of its own which contains only the null value:

/

let age = null;
The ‘null’ data type basically defines a special value that represents ‘nothing’,

’empty’, or ‘value unknown’. **Undefined** Just like null, Undefined makes its own type. The meaning of undefined is ‘value is not assigned’.

let x;
console.log(x); // undefined **Symbol:**

Symbols are new primitive built-in object types introduced as part of ES6. Symbols return unique identifiers that can be used to add unique property keys

to an object that won’t collide with keys of any other

code that might add to the object. They are used as object properties that cannot be recreated. It basically helps us to enable encapsulation or information hiding.

let symbol1 = Symbol("") let symbol2 = Symbol("")

/ Each time Symbol() method
// is used to create new global Symbol

```
console.log(symbol1 == symbol2); // False
```

BigInt:

BigInt is a built-in object in JavaScript that provides a

way to represent whole numbers larger than 253-1.

• **Object:** It is the most important data type and forms the building blocks for modern JavaScript.

JavaScript Non-Primitive Data Types Examples:

Examples: Object: JavaScript objects are fundamental data structures used to store collections of data. They consist of key-value pairs and can be created using curly braces {} or the new keyword. Understanding objects is crucial, as everything in JavaScript is essentially an object.

Using the “object constructor” syntax:

```
let person = new Object();
```

JavaScript. Variables are used to store reusable values. The values of the variables are allocated using the assignment operator(“=”).

Basic rules to declare a variable in JavaScript:

These are case-sensitive.

Basic rules to declare a variable in JavaScript:

These are case-sensitive.

- Can only begin with a letter, underscore(“_”) or “$” symbol.

- It can contain letters, numbers, underscore, or “$” symbol.

- A variable name cannot be a reserved keyword. **JavaScript** is a dynamically typed language so the type of variables is decided at runtime. Therefore there is no need

to explicitly define the type of a variable. We can declare variables in JavaScript in three ways:

- **JavaScript var keyword**

- JavaScript let keyword

- JavaScript const keyword
  Note: In JavaScript, variables can be declared automatically. •
  Syntax:
  // Declaration using var var hello = "Hello hello" // Declaration using let let $ = "Welcome"
  // Declaration using const
  const _example = "hello"
  All three keywords do the basic task of declaring a variable but with some differences Initially, all the variables in JavaScript were written using the var

keyword but in ES6 the keywords let and const were introduced.

Example 1: In this example, we will declare variables using var.

var a = "Hello"; **var** b = 10; **var** c = 12;

console.log(a); console.log(b); console.log(c);

console.log(d); **Output**

Hello 10 12
22

Example 2: In this example, we will declare variables using let.

let a = "Hello learners" **let** b = "joining"; **let** c = " 12"; **let** d = b + c;

console.log(a); console.log(b); console.log(c); console.log(d); **Output**

Hello learners joining
12 joining 12

Example 3: In this example, we will

declare the variable using the const keyword.

const a = "Hello learners" console.log(a);

const b = 400; console.log(b);

const c = "12"; console.log(c);

// Can not change a value for a constant c = "new" console.log(c)

Output:

and const are block scoped whereas var is function scoped.

When to Use var, let, or const

- We declare variables using const if the value should not be changed.

- We use const if the type of the variables should not be changed such as working with Arrays and objects.

- We should use let if we want mutable value or we cannot use const. We use var only if we support old browser.

  JavaScript Case Sensitive:
  JavaScript Identifiers are case-sensitive.

Example: Both the variables firstName and

firstname are different from each other. **let** firstName = "hello"; **let** firstname = 100; console.log(firstName); console.log(firstname);

Output

hello 100

JavaScript Camel Case:

In JavaScript Camel case is preferred to name an identifier.

Example:

let firstName • let lastName

JavaScript Operators:

JavaScript **operators** are symbols used to operate the operands. Operators are used to perform specific mathematical and logical computations on operands. **JavaScript Operators:** There are various operators supported by JavaScript.

• JavaScript Arithmetic Operators

- JavaScript Assignment Operators
- JavaScript Comparison Operators
- JavaScript Logical Operators
- JavaScript Bitwise Operators
- JavaScript Ternary Operators
- JavaScript Comma Operators
- JavaScript Unary Operators
- JavaScript Relational Operators
- JavaScript BigInt Operators

- JavaScript String Operators

JavaScript Arithmetic Operators:

JavaScript Arithmetic Operators perform arithmetic operations: addition (+), subtraction (-), multiplication (*), division (/), modulus (%), and exponentiation (**).
Addition(+) Arithmetic Operator in JavaScript: JavaScript arithmetic addition operator is capable of performing two types of operation that is addition and concatenation. It is used to perform some on the number or concatenate Strings.
Syntax:
a+b

Syntax:

a-b

Return Type: It returns a number after subtracting

the operators. console.log(200-100); //100

Multiplication(*) Arithmetic Operator in JavaScript:

JavaScript arithmetic multiplication operator is used to find the product of operands. Like if we want the multiplication of any two numbers then this operator can be useful.

Syntax:

a*b

Return: It returns the product of the operands. console.log(100*20); //2000

Division(/) Arithmetic Operator in JavaScript:

JavaScript arithmetic division operator is used to find the quotient of operands. The left operator is treated as a dividend and the right operator is treated as a divisor.

Syntax:

a/b

Return Type: It returns the quotient of the operands console.log(100/20); // 5

Modulus(%) Arithmetic Operator in JavaScript:

JavaScript arithmetic modulus operator is used to return the remainder of the operands. Here also the left operator is the dividend and the right operator is the divisor.

Syntax:

a%b

Return Type: Remainder of the operands. console.log(100%23); //8

Exponentiation(**) Arithmetic Operator in JavaScript:

JavaScript exponentiation(**) operator in JavaScript is represented by “**” and is used to find the power of the first operator

raised to the second operator. This operator is equal to Math.pow() but makes the code simpler and can even accept BigInt primitive data type. The exponentiation operator is right associative which means that x**y**z will give the same result as x**(y**z).

Syntax:

a**b

Return: It returns the result of raising the first operand to the power of the second operand.

let a=4; let b=3; let c=-2

64 81 16

0.0625

Increment(+ +) Arithmetic Operator in JavaScript:

JavaScript increment(+ +) operator is used to increase the value of the variable by one. The value returned from the operand depends on whether the

increment operator was on the left(prefix increment) or right(postfix increment). If the operator is used before the operand then the value is increased by one and then returned but if the operator is after the operand then the value is first returned and then incremented.

The increment operator can only be used on references that is the operator can only be applied to variable and object properties. Also, the increment operator cannot be chained

Syntax:

a++ OR ++a

let x = 10; console.log(x++); console.log(x); 10 11

let x = 10; console.log(++x); console.log(x); 11 11

Decrement(–) Arithmetic Operator in JavaScript:

JavaScript decrement operator is used to

decrease the value of the variable by one. The value returned from the operand depends on whether the decrement operator was on the left(prefix decrement) or right(postfix decrement). If the operator is used before the operand then the value is decreased by one and then returned but if the operator is after the operand then the value is first returned and then decremented.

The decrement operator can only be used on references that is the operator can only be applied to variable and object properties. Also, the decrement operator cannot be chained.

Syntax:

--a OR a--

let x = 10; console.log(x--); console.log(x);
10 9

let x = 10; console.log(--x); console.log(x);
99

JavaScript Assignment Operators:

The assignment operation evaluates the assigned value. Chaining the assignment operator is possible in

order to assign a single value to multiple variables.

Addition Assignment (+=) Operator in Javascript: JavaScript Addition assignment operator(+=**)** adds a value to a variable, The Addition Assignment (+ =) Sums up left and right operand

values and then assigns the result to the left operand. The two major operations that can be performed using this operator are the addition of numbers and the concatenation of strings.

Syntax:

a += b

//concatenation of two number let x = 2;

```
let y = 5;
console.log(x += y); //7
```

Subtraction Assignment(-=) Operator in Javascript:

The **Subtraction Assignment Operator(-=)** is used to subtract a value from a variable. This operator subtracts the value of the right operand from a variable and assigns the result to the variable, in other words, allows us to decrease the left variable from the right value.

Syntax:

```
a -= b
```

```
let number = 9;
number -= 5; console.log(number);//4
```

Multiplication Assignment(*=) Operator in JavaScript: Multiplication Assignment Operator(*=) in JavaScript is used to

```
let number = 9;
```

number -= 5; console.log(number);//4

Multiplication Assignment(*=) Operator in JavaScript: Multiplication Assignment Operator(*=) in JavaScript is used to

multiply two operands and assign the result to the right operand.

Syntax:

variable1 *= variable2
// variable1 = variable1 * variable2 let number = 5;
number *= 2; console.log(number);
// Expected output:10

Division Assignment(/=) Operator in JavaScript:

JavaScript **Division Assignment Operator** in JavaScript is represented by **“/=”.** This operator is used to divide the value of the variable by the value of the right operand and that result is going to be assigned to the variable. This can also be explained as dividing the value of the

variable by an expression and assigning that result to the variable.

Syntax:

a /= b
or a=a/b

let x = 18;

x /= 3; console.log(x); x /= 2; console.log(x); x /= 0; console.log(x); 6 3 Infinity

JavaScript Comparison Operators:

Comparison operators are mainly used to perform the logical operations that determine the equality or difference between the values.

Equality(==) Comparison Operator in JavaScript:

In JavaScript, the Comparison Equality Operator is used to compare the values of the operand. The comparison operator

returns true only if the value of two operands are equal otherwise it returns false. It is also called a loose equality check as the operator performs a type conversion when comparing the elements. When the comparison of objects is done then only their reference is checked even if both objects contain the same value.

Syntax:

a==b

```
let a = 1;

let b = 1;
let c = new String("Hello);
let d = new String("Hello");
let e = "Hello";
console.log(a==b);
console.log(c==d);
console.log(c==e);
```

true
false
true

Strict Equality(===) Comparison

Operator in JavaScript:

JavaScript Strict Equality Operator is used to compare two operands and return true if both the value and type of operands are the same. Since type conversion is not done, so even if the value stored in operands is the same but their type is different the operation will return false.

Syntax:

a===b

let a = 2, b=2, c=3; let d = {name:"Ram"}; let e = {name:"Ram"}; let f = e; console.log(a===b);

console.log(a===c); console.log(f===e);
true
false

true

console.log(a===c);

compare two operands and return true if the left

console.log(f===e); true
false

true

Inequality(!=)
Comparison Operator in JavaScript:

JavaScript Inequality operator is used to compare two operands and returns true if both the operands are unequal it is basically the opposite of the Equality Operator. It is also called a loose inequality check as the operator performs a type conversion when comparing the elements. Also in the case of object comparison, it only compares the reference of the objects.

Syntax:

a != b

let a = 1;
let b = 2;
let c = new String("Hello");
let d = new String("Hello");

```
let e = "Hello";
console.log(a!=b);
console.log(c!=d);
console.log(c!=e);
```

true
true
false

Greater than(>) Comparison Operator in JavaScript:

JavaScript Greater Than(>) Operator is used to

operand has a higher value than the right operator.

Syntax:

Greater than(>) Comparison Operator in

compare two operands and return true if the left

JavaScript:

JavaScript Greater Than(>) Operator is used to

operand has a higher value than the right operator.

Syntax:

a>b

console.log("3">2); console.log("2">3);
true false

Less Than(<) Comparison Operator in JavaScript:

JavaScript Less Than(<) Operator is used to compare two operands and return true if the left operand has a lesser value than the right operator. The values are converted to equal primitive types before conversion If both the values are strings the comparison is done on the basis of their Unicode. Boolean values like true and false are converted to 1 and 0 respectively.

Syntax:

a<b

console.log("3"<2); console.log("2"<3);

false true

Greater Than or Equal(>=) Comparison Operator in JavaScript:

JavaScript Greater Than or Equal Operator(>=) is used to compare two operands and return true if the left operand has a value greater than or equal to the right operand. The algorithm used in the comparison

is similar to that of less than comparison the only difference is that the values are swapped and equal values are checked before returning a boolean.

JavaScript Greater Than or Equal Operator(>=) is

used to compare two operands and return true if the left operand has a value greater than or equal to the right operand. The algorithm used in the comparison

f

is similar to that of less than comparison

the only difference is that the values are swapped and equal values are checked before returning a boolean.

Syntax:

a>=b

console.log("3">=2); console.log("2">=3);
console.log(true>=false);
console.log("3">="2"); console.log(3>=2);

true

alse true true true

Less Than or Equal(<=) Comparison Operator in JavaScript:

JavaScript Less Than or Equal(<=) to the operator is used to compare two operands and return true if the left operand is smaller or equal to the right operand. The algorithm used for the comparison is the same as that of less than operator but equal condition is also checked

Syntax:

a<=b

```
console.log("3"<=2);
console.log("2"<=3);
console.log(true<=false);
console.log("3"<="2");
console.log(3<=2);
```

false true false false false

```
console.log("hello world")
console.log(1)
console.log(true)
console.warn('This is a warning message')
console.error('There is an error program')
console.clear() //Singleline comment

/* Multiple line comment */

var a="hi" //Assignment operator let b="hello"
console.log(a)
console.log(b)

const c="bye" console.log(c)

// Arithmentic operators

let a1=10,b1=5
console.log(a1+b1) //Addition operator console.log(a1-
b1) //subtraction operator console.log(a1*b1)//
multiplication operator console.log(a1/b1) //division
operator console.log(++a1)//increment operator
console.log(--b1)//decrement operator
console.log(a1%b1)//Remainder operator

//Comparison operators
let a2=4,b2=4
console.log(a2==b2) //Equality operator
```

```
let a3=4,b3=5
console.log(a3!=b3) //Inequality operator
console.log(a3>b3) //greater thsn operator
console.log(a3<b3) //less than operator
console.log(a3<=b3) //less than or equalsto operator let
a4=6,b4=6
console.log(a4>=b4) //greater or equals to operator

let a5='3',a6=3 // different datatypes with same value
console.log(a5==a6) // Equality operator
console.log(a5===a6) // Strict equality

let b5=3,b6=4,b7=8
console.log(b5<b6 && b6<b7) //Logical and
console.log(b5>b6 && b6<b7) //Logical and
console.log(b5>b6 && b6>b7) //Logical and
console.log(b5<b6 llb6<b7) //Logical or
```

JavaScript Logical Operators:

JavaScript Logical Operators perform logical operations: AND (&&), OR (||), and NOT (!), evaluating expressions and returning boolean values.

AND(&&) Logical Operator in JavaScript:

JavaScript Logical And(&&) Operator or Logical Conjunction Operator operates on a set of operands and returns true only if all the operands are true otherwise returns

false. It operates the operands from left to right and returns false whenever the first falsy value is encountered.

The Logical AND(&&) Operator can also be used on

non-boolean values also. AND operator has higher

precedence than the OR operator.

Syntax:

a&&b

console.log(true && false); console.log(true && true); console.log(1 && 0); console.log(1 && 2); console.log("1" && true); console.log("0" && true); false

true
0
2
true
true

OR(||) Logical Operator in JavaScript:

JavaScript logical OR(||) Operator or Logical disjunction Operator operates on a set of operands and returns true even if one of the operands is true. It evaluates the operands from left to right and returns true whenever it encounters the first truthy value otherwise false.

The Logical OR Operator can be used on non- boolean values but it has a lower precedence than the AND operator.

Syntax:

allb

console.log(true || false); console.log(false || false); console.log(1 || 0);

console.log(1 || 2);

Syntax:

allb

console.log(true || false); console.log(false || false); console.log(1 || 0); console.log(1 || 2); console.log("1" || true);

console.log("0" ll true); true false 1

110

NOT(!) Logical Operator inJavaScript:

JavaScript NOT Operator can be used to find the flipped value of a boolean. It can be used to convert a true value to a false and vice-versa. This NOT operator can also be used on non-booleans to convert them to the reverse of their actual boolean value. A NOT operator can be used with another NOT operator to get the original value back.

Syntax:

!a

console.log(!true); console.log(!false); console.log(!"1"); console.log(!""); console.log(!null); false

true false true true

JavaScript Loops:

JavaScript Loops are powerful tools for

performing repetitive tasks efficiently. Loops in JavaScript execute a block of code again and again while the condition is true.

For-loop:

for (initialization; testing condition; increment/ decrement) {

statement(s) }

For example, suppose we want to print “Hello World” 5 times. This can

be done using JS Loop easily. In Loop, the statement needs to be

Hello World! Hello World! Hello World!

JavaScript while Loop:

The JS while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax:

while (boolean condition) { loop statements...

}
// JavaScript code to use while loop let val = 1;
while (val < 6) {

console.log(val); val += 1; }

12345

JavaScript do-while Loop:

do {

while (condition);

Statements... }

let test = 1; do {

console.log(test);

test++;
} while(test <= 5) 12345

JavaScript for-in Loop:

JS for-in loop is used to iterate over the properties of an object. The for- in loop iterates only over those keys of an object which have their enumerable property set to “true”.

Syntax

for(let variable_name in object_name) { // Statement

}
let myObj = { x: 1, y: 2, z: 3 }; for (let key in myObj) {

console.log(key, myObj[key]); }

x1

execution of the loop or switch statement when the condition is true.

Syntax

break;

for (let i = 1; i < 6; i++) { if (i == 4)

break;

console.log(i); }

123

JavaScript Continue Statement:

JS continue statement is used to break the iteration of the loop and follow with the next iteration. The break in iteration is possible only when the specified condition going to occur. The major difference between the continue and break statement is that the break statement breaks out of the loop completely while continue is used to break one statement and iterate to the next statement.

Syntax

continue;

for (let i = 0; i < 11; i++) { if (i % 2 == 0)

continue;

console.log(i); }

13579

for(let x=10;x>=5;x--){ //for-loop console.log("Value of x is:" +x)

}

let y for(y=5;y<=8;y++){

console.log("Value of y is:"+y)

}
let x1=6
while(x1<=5){ //while-loop--will execute only if condition will satisfy

console.log(" Value of x1 is:" +x1)

x1++ }

let y1=10 while(y1>=6){

console.log("Value of y1 is:" +y1)

y1--; }

let z=4//do-while --you will get output atleast once irrespective of condition being satisfied.
do{

console.log("Value of z is:" +z)

z++ }while(z<=7)

for(let i=0;i<9;i++){ //continue if(i==7) continue; console.log(i)

}

let a=20
if(a<10){ //If statement can be executed with any condition and operator

```
condition and operator

 console.log("If statement is executed") } else{ //If-else
 statement

 console.log("Else statement is being executed") }

 let b=10
 if(a+b>0){ //elseif statement to check multiple possible
 conditions

 console.log("b is a positive number") } else if(b<0){

 console.log("b is a negative number")
} else {
console.log("b is zero")

}
console.log(a+b) // Adding two different variables

let marks=85,subjects switch(true){

case marks>65: subjects="HTML"; break;

case marks>85: subjects="CSS"; break;

case marks=90: subjects="JS";

break; }

console.log(`Subject that is being taught is ${subjects}`)

let subject="javascript"
console.log(`My name is Bhargavi and I'm studying $
{subject}` )

let a1=5,b1=10
console.log("addition of a1 and b1 is" +" "+ (a1+b1))
console.log(`Addition of a1 and b1 is ${a1+b1}`)

let x for(x=5;x<=4;x++){
```

```
console.log("Value of x is:" +x) }
```

JavaScript Strings:

Strings are essential in any programming language, including JavaScript.

What is String in JavaScript?

JavaScript String is a sequence of characters, typically used to represent text. It is enclosed in single or double quotes and supports various methods for text manipulation.

You can create JavaScript Strings by enclosing text in single or double quotes. String literals and the String constructor provide options. Strings allow dynamic manipulation, making it easy to modify or extract elements as needed.

Basic Terminologies of JavaScript String:

- **String: A sequence of characters enclosed in single (‘ ‘) or double (” “) quotes.**

- Length: The number of characters in a string, obtained using the length property.

- Index: The position of a character within a string, starting from 0.

- Concatenation: The process of combining two or more strings to create a new one.

- Substring: A portion of a string, obtained by extracting characters between specified indices.

Declaration of a String: 1. Using Single Quotes:

Single Quotes can be used to create a

string in JavaScript. Simply

enclose your text within single quotes to declare a string.

Syntax:

let str = 'String with single quote';

2. Using Double Quotes:

Double Quotes can also be used to create a string in JavaScript. Simply

enclose your text within double quotes to declare a string.

Syntax:

let str = “String with double quote”;

3. String Constructor:

You can create a string using the String Constructor. The String Constructor is less common for direct string creation, it provides additional methods for manipulating strings. Generally, using string literals is preferred for simplicity.

```
let str = new String('Create String with String Constructor');

console.log(str);
```

4. Using Template Literals (String Interpolation):

You can create strings using Template Literals. Template literals allow you to embed expressions within backticks (`) for dynamic string creation, making it more readable and versatile.

Syntax:

```
let str = 'Template Litral String';
let newStr = `String created using ${str}`;
```

5. Empty String:

You can create an empty string by assigning either single or double

quotes with no characters in between.

Syntax:

```
// Create Empty String with Single Quotes
```

let str1 = '';
// Create Empty String with Double Quotes let str2 = "";

6. Multiline Strings (ES6 and later):

You can create a multiline string using backticks (“) with template literals. The backticks allow you to span the string across multiple lines, preserving the line breaks within the string.

Syntax:

let str = ` This is a multiline

string`;

Basic Operations on JavaScript Strings:

1. Finding the length of a String:

You can find the length of a string using the length property.

Length: " + len); String Length: 10

2. String Concatenation:

You can combine two or more strings using + Operator.

Example:

```
let str1 = 'Java';
let str2 = 'Script';
let result = str1 + str2;
console.log("Concatenated String: " +
result); Concatenated String: JavaScript
```

3. Escape Characters:

We can use escape characters in string to add single quotes, dual quotes, and backslash.

Syntax:

```
\' - Inserts a single quote
\" - Inserts a double quote
\\ - Inserts a backslash
const str1 = "\'hello\' is a learning portal";
const str2 = "\"hello\" is a learning portal";
```

const str3 = "\\hello\\ is a learning portal";

console.log(str1); console.log(str2); console.log(str3);

'hello' is a learning portal "hello" is a learning portal \hello\ is a learning portal

4. Breaking Long Strings:

We will use a backslash to break a long string in multiple lines of code.

const str = "'for' is \ a learning portal";

console.log(str);
'for' is a learning portal

5. Find Substring of a String:

We can extract a portion of a string using the substring() method.

let str = 'JavaScript Tutorial'; **let** substr = str.substring(0, 10);

console.log(substr); JavaScript

6. Convert String to Uppercase and Lowercase:

Convert a string to uppercase and lowercase using toUpperCase() and

toLowerCase() methods.
let str = 'JavaScript';
let upperCase = str.toUpperCase(); **let** lowerCase = str.toLowerCase();

console.log(upperCase);
console.log(lowerCase); JAVASCRIPT javascript

let str="string1" //double quotes let str1='string2' //single quotes console.log(str) console.log(str1)

let str2=new String('Creating string with string constructor') // create a new string
console.log(str2)

let str3='' //with single quotes or double quotes --empty string

let str="string1" //double quotes let str1='string2' //single quotes console.log(str) console.log(str1)

let str2=new String('Creating string with string constructor') // create a new string
console.log(str2)

let str3='' //with single quotes or double quotes --empty string console.log(str3)

let str4=` //Multiline string I'm Bhargavi
and

I'm teaching JS` console.log(str4)

// String methods

let str5="INDIA"
//Method to find the length of the string - str.length
console.log(str5.length)

let str6="Movie"
let str7=" Actors" //Concatenate two strings
console.log(str6+str7)

let str8="LEVELUP"
let substr=str8.substring(0,5) //To extract substring from main string --- str.substring
console.log(substr)

let str9="INSTITUTE"
let l=str9.toLowerCase(); // to change the string to lowercase console.log(l)

let str10="levelup"
let l1=str10.toUpperCase(); // to change the string to uppercase console.log(l1)

let str11="JAVASCRIPT"
let search=str11.indexOf('S') //To find the index position of a string console.log(search)

let str12="I'm learning HTML"
let newStr=str12.replace('HTML','JAVASCRIPT') //To replace my old string to

console.log(newStr)

let str13=" Hi"
let new1=str13.trim(); //To trim unneccessary white spaces

new string

```
console.log(new1)

console.log(search)

let str12="I'm learning HTML"
let newStr=str12.replace('HTML','JAVASCRIPT') //To
replace my old string to

console.log(newStr)

let str13=" Hi"
let new1=str13.trim(); //To trim unneccessary white spaces
console.log(new1)

let str14=" Helloo Bye " let new2=str14.trim()
console.log(new2)

let str19=" Helloo Bye " let newStr4=str19.replace('
console.log(newStr4.trim())

',' ') //To replace my old string to new string

let str15="Thank you"
let str16="welcome"
let new3=str16.toUpperCase() let new4=str15+new3
console.log(new4)

let str17="javascript"
let new5=str17.toUpperCase() let
search1=new5.indexOf('S') console.log(search1)

let str18="I'm learning HTML"
let newStr1=str18.replace('html','JAVASCRIPT') //To
replace my old string to new string
console.log(newStr1)

let str20="Javascript"
let char=str20.charAt(4) // To find the character at
particular index position console.log(char)
```

```
let str21="Heloooo"
let str22="Helo"
console.log(str21==str22) //To compare two strings --
return boolean

let str23=new String("Helo") console.log(str21==str23)

console.log(str21.localeCompare(str22)) // To give
difference of two strings --return will 0,1,-1
```

7. String Search in JavaScript:

new string

Find the index of a substring within a string using

```
console.log(str21==str23)

console.log(str21.localeCompare(str22)) // To give
difference of two strings --return will 0,1,-1
```

7. String Search in JavaScript:

Find the index of a substring within a string using indexOf() method.

```
let str = 'Learn JavaScript at hello';
let searchStr = str.indexOf('JavaScript');

console.log(searchStr); //6
```

8. String Replace in JavaScript:

Replace occurrences of a substring with

another using replace() method.

let str = 'Learn HTML at hello';
let newStr = str.replace('HTML', 'JavaScript'); console.log(newStr); //Learn JavaScript at hello **9. Trimming Whitespace from String:**

Remove leading and trailing whitespaces using trim() method.

let str = ' Learn JavaScript '; **let** newStr = str.trim(); console.log(newStr); //Learn JavaScript **10. Access Characters from String:**

Access individual characters in a string using bracket notation and

charAt() method.

let str = 'Learn JavaScript'; **let** charAtIndex = str[6]; console.log(charAtIndex); charAtIndex = str.charAt(6); console.log(charAtIndex);

compare strings

such as the equality operator and another like localeCompare() method. **let** str1 = "John";

let str2 = **new** String("John");
console.log(str1 == str2);

console.log(str1.localeCompare(str2));
true 0

10. Passing JavaScript String as Objects:

We can create a JavaScript string using the new keyword.

const str = **new** String("for");
console.log(str);

[String: 'for']

Are the strings created by the new keyword is same as normal strings?
No, the string created by the new keyword is an object and is not the same as normal strings.

const str1 = new String("for"); const str2 = "for"; console.log(str1 == str2);

console.log(str1 === str2); true

false

JavaScript Arrays:

objects, and even other arrays. Arrays in JavaScript are zero- indexed i.e. the first element is accessed with an index 0, the second element with an index

of 1, and so forth.

Basic Terminologies of JavaScript Array:

- **Array: A data structure in JavaScript that allows you to store multiple values in a single variable.**

- Array Element: Each value within an array is called an element. Elements are accessed by their index.

- Array Index: A numeric representation

that indicates the position of an element in the array. JavaScript arrays are zero-indexed, meaning the first element is at index 0.

- Array Length: The number of elements in an array. It can be retrieved using the length property
  Declaration of an Array:
  There are basically two ways to declare an array i.e. Array Literal and Array Constructor. Creating an Array using Array Literal:
  Creating an array using array literal involves using square brackets [] to define and initialize the array. This method is concise and widely preferred for its simplicity.
  Syntax:
  let arrayName = [value1, value2, ...];
  •

// Creating an Empty Array • let names = [];
• console.log(names);
•

Syntax:

let arrayName = [value1, value2, ...]; •

// Creating an Empty Array

- let names = [];

- console.log(names); •
  // Creating an Array and Initializing with Values

- let courses = ["HTML", "CSS", "Javascript", "React"]; •

console.log(courses);
[]
['HTML', 'CSS', 'Javascript', 'React']

Creating an Array using Array Constructor (JavaScript new Keyword):

The “Array Constructor” refers to a method of creating arrays by invoking the Array constructor function. This approach allows for dynamic initialization and can be used to create arrays with a specified length or elements.

Syntax:

```
let arrayName = new Array();
// Declaration of an empty array
// using Array constructor
let names = new Array();
console.log(names);
// Creating and Initializing an array with values
let courses = new Array("HTML", "CSS", "Javascript", "React");
```

arr[2] = 30;
console.log(arr);
[]
['HTML', 'CSS', 'Javascript', 'React']
[10, 20, 30]

Basic Operations on JavaScript Arrays:

1. Accessing Elements of an Array

Any element in the array can be accessed using the index number. The index in the arrays starts with 0.

// Creating an Array and Initializing with Values
let courses = ["HTML", "CSS", "Javascript", "React"]; // Accessing Array Elements console.log(courses[0]);
console.log(courses[1]);
console.log(courses[2]);
console.log(courses[3]);

HTML CSS Javascript

2. Accessing the First Element of an Array:

The array indexing starts from 0, so we can access first element of array

using the index number.
// Creating an Array and Initializing with Values let courses = ["HTML", "CSS", "JavaScript", "React"]; // Accessing First Array Elements
let firstItem = courses[0];
console.log("First Item: ", firstItem);
First Item: HTML

3. Accessing the Last Element of an Array:

We can access the last array element using [array.length – 1] index number.

// Creating an Array and Initializing with Values

let courses = ["HTML", "CSS", "JavaScript", "React"]; // Accessing Last Array Elements let lastItem = courses[courses.length - 1];

courses[courses.length - 1];
console.log("First Item: ", lastItem);

First Item: React

---

4. Modifying the Array Elements:

Elements in an array can be modified by assigning a new value to their

corresponding index.

// Creating an Array and Initializing with Values

let courses = ["HTML", "CSS", "Javascript", "React"];

console.log(courses);

courses[1]= "Bootstrap";

console.log(courses);

['HTML', 'CSS', 'Javascript', 'React']
['HTML', 'Bootstrap', 'Javascript', 'React']

5. Adding Elements to the Array:

Elements can be added to the array using methods like push() and unshift()

using methods like push() and unshift().

```
// Creating an Array and Initializing with
Values
let courses = ["HTML", "CSS",
"Javascript", "React"]; // Add Element to
the end of Array
courses.push("Node.js");
// Add Element to the beginning
courses.unshift("Web Development")
console.log(courses);
```

```
[ 'Web Development', 'HTML', 'CSS',
'Javascript', 'React', 'Node.js' ]
```

6. Removing Elements from an Array:

Remove elements using methods like pop(), shift(), or splice().

```
// Creating an Array and Initializing with
Values

let courses = ["HTML", "CSS",
"Javascript", "React", "Node.js"];

console.log("Original Array: " + courses);

// Removes and returns the last element
```

```
let lastElement = courses.pop();

console.log("After Removed the last elements: " + courses);

// Removes and returns the first element
let firstElement = courses.shift();

console.log("After Removed the First elements: " + courses);

// Removes 2 elements starting from index 1 courses.splice(1, 2);

console.log("After Removed 2 elements starting from index 1: " + courses);
```

length property. // Creating an Array and Initializing with Values
let courses = ["HTML", "CSS", "Javascript", "React", "Node.js"];

let len = courses.length;
console.log("Array Length: " + len); Array Length: 5

8. Increase and Decrease the Array Length:

We can increase and decrease the array length using the JavaScript

length property.
// Creating an Array and Initializing with Values
let courses = ["HTML", "CSS", "Javascript", "React", "Node.js"]; // Increase the array length to 7
courses.length = 7;
console.log("Array After Increase the Length: ", courses);

courses.length = 2;
console.log("Array After Decrease the Length: ", courses)

Array After Increase the Length: ['HTML', 'CSS', 'Javascript', 'React', 'Node.js', <2 empty items>] Array After Decrease the Length: ['HTML', 'CSS']

```
let array=[] // Empty string console.log(array)

let arr1=[1,2,3,4] //Array with integers console.log(arr1)

let arr2=["Hi","Bye"] //Array with strings console.log(arr2)

let arr3=new Array("Html","CSS") // Creation of array by invoking array constructor
console.log(arr3)

//Array Methods

let arr4=["HTML","CSS","JS"] //Accessing elements
console.log(arr4[1])
console.log(arr4[0]) // To access first element

console.log(arr4[arr4.length-1]) // To access last element

arr4[1]="ReactJS" // To replace a particular string
console.log(arr4)

let arr5=["Vijay","Ajay","Sanjay"]
arr5.push("React")
console.log(arr5) // To add element to your array-- end of an array

arr5.unshift("Hiiiii")
console.log(arr5) // To add element to beginning of an array
```

```
arr5.pop(); console.log(arr5)

arr5.shift() console.log(arr5)

arr5.splice(2) console.log(arr5)

// To remove last element of an array
// To remove element from the beginning

let arr6=["hey","bye","zero","one"]
console.log(arr6.length) // To find the length of an array
```

9. Iterating Through Array Elements:

We can iterate array and access array elements using for and forEach loop.

Example: It is the example of for loop.

```
// Creating an Array and Initializing with Values
let courses = ["HTML", "CSS", "JavaScript", "React"] // Iterating through for loop
for (let i = 0; i < courses.length; i++) {
```

10. Array Concatenation:

Combine two or more arrays using the concat() method. Ir returns new

array containing joined arrays elements.

```
// Creating an Array and Initialising with
```

Values
let courses = ["HTML", "CSS", "JavaScript", "React"]; let otherCourses = ["Node.js", "Expess.js"];

// Concatenate both arrays
let concateArray = courses.concat(otherCourses);

console.log("Concatenated Array: ", concateArray);

Concatenated Array: ['HTML', 'CSS', 'JavaScript', 'React', 'Node.js', 'Expess.js']

11. Conversion of an Array to String:

We have a builtin method **toString()** to converts an array to a string. // Creating an Array and Initializing with Values
let courses = ["HTML", "CSS", "JavaScript", "React"];

// Convert array ot String
console.log(courses.toString());
HTML,CSS,JavaScript,React

type of an array. It

let courses = ["HTML", "CSS", "JavaScript", "React"];

key_name : value,

returns “object” for arrays.
// Creating an Array and Initializing with Values

// Check type of array console.log(typeof courses); object

Difference Between JavaScript Arrays and Objects:

- JavaScript arrays use indexes as numbers.

- objects use indexes as names.
  When to use JavaScript Arrays and Objects?

- Arrays are used when we want element names to be numeric.

- Objects are used when we want element names to be strings.
  •

Objects in Javascript:

Objects, in JavaScript, are the most important data type and form the building blocks for modern JavaScript. These objects are quite different from JavaScript's primitive data types (Number, String, Boolean, null, undefined, and symbol) in the sense that these primitive data types all store a single value each (depending on their types).

Syntax:

new Object(value)

• Object(value)
• let object_name = { •

on their types).

Syntax:

new Object(value)

.

• Object(value)
• let object_name = { •

key_name : value, •

.. }

Note:- Object() can be called with or without new. Both create a new object.

const o = new Object(); o.foo = 42;

console.log(o);
// { foo: 42 }
// JavaScript code demonstrating a simple object let school = {

name: 'Vivekananda School', location: 'Delhi', established: '1971', displayInfo: function () {

console.log(`${school.name} was established in ${school.established} at ${school.location}`);

}}

school.displayInfo();

Vivekananda School was established in 1971 at Delhi

• Objects are more complex and each object may contain any combination of these primitive data- types as well as reference data-types.

• An object is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don't actually store the value.

• Loosely speaking, **objects in JavaScript may be defined as an**

unordered collection of related data, of primitive or reference

types, in the form of "key: value" pairs. These keys can be variables or functions

and are called properties and methods, respectively, in the context of an object.

An object can be created with figure brackets {...} with an optional list of properties. A property is a “key: value” pair, where a key is a string or a symbol (also called a “property name”), and the value can be anything including numbers, strings, booleans, functions, arrays, or even other objects.

Iterating over all keys of an object:

To iterate over all existing enumerable keys of an object, we may use the **for...in** construct. It is worth noting that this allows us to access only those

properties of an object which are enumerable (Recall

But, enumerable properties inherited from somewhere can also be accessed using the **for...in** construct

Example: Below is the example: **let** person = {

gender: "male" }

let person1 = Object.create(person);
person1.name = "Adam";
person1.age = 45; person1.nationality = "Australian";

for (**let** key **in** person1) {
// Output : name, age, nationality // and gender console.log(key);

}

name
age nationality gender

Deleting Properties:

To Delete a property of an object we can make use of the delete operator. An example of its usage has been listed below.

```
let obj1 = {
```

```
propfirst : "Name" }

// Output : Name
console.log(obj1.propfirst); delete
obj1.propfirst

// Output : undefined
console.log(obj1.propfirst);

// Simple object

let Student={ roll: 12,

name: "siri", subject:"JS", office:"levelup"

}
console.log(Student) //Entire data
console.log(Student.name) //only name
console.log(Student.office) //only office output

//Complex objects let Student1={

roll1:14, name1:{

firstname: "Bhargavi", lastname:"GVS",
middlename:"Siri"

}, subject1: {

sub1:"html", sub2:"css", sub3:"js",

},

office1:"levelup" }
```

```
console.log(Student1) console.log(Student1.subject1)
console.log(Student1.subject1.sub2)

delete Student1.office1; //delete any property in object
console.log(Student1)

for(let x in Student1){ //for..in loop--- it is used to iterate
over properties console.log(x)

}
```

Functions in JavaScript

Java Script functions are essential for organizing code and executing specific tasks. They contain sets of instructions that run when triggered by events or actions. In this article, we'll explore the syntax, parameters, return values, and execution contexts of JavaScript functions. Through practical examples, we'll provide a clear understanding of how to use functions effectively in web development.

A JavaScript function is executed when "something" invokes it (calls it).

Example 1: A basic javascript function, here we create a function that divides the

1st element by the second element.

```
function myFunction(g1, g2) { return g1 / g2;

}
const value = myFunction(8, 2); // Calling the
function console.log(value);
```

Output:

4

Syntax: The basic syntax to create a function in JavaScript is shown below.

```
function functionName(Parameter1,
Parameter2, ...) {

// Function body }
```

To create a function in JavaScript, we have to first use the keyword function, separated by the name of the function and parameters within parenthesis. The part of the function inside the curly braces {} is the body of the function.

In javascript, functions can be used in the same way as variables for assignments, or calculations.

Function Invocation:

- Triggered by an event (e.g., a button click by a user).

- When explicitly called from JavaScript code.

- Automatically executed, such as in self-invoking functions.

Function Definition:

Before, using a user-defined function in JavaScript we have to create one. We can use the above syntax to create a function in JavaScript. A function definition is sometimes also termed a function declaration or

function statement. Below are the rules for creating a

function in JavaScript:

- Every function should begin with the keyword function followed by,

- A user-defined function name that should be unique,

- A list of parameters enclosed within parentheses and separated by commas,

- A list of statements composing the body of the function enclosed within curly braces {}.
  Example 2: This example shows a basic declaration of a function in javascript.
  function calcAddition(number1, number2)
  { **return** number1 + number2;
  } console.log(calcAddition(6,9));
  Output
  15

In the above example, we have created a function named calcAddition,

- This function accepts two numbers as parameters and returns the addition of these two numbers.

- Accessing the function with just the function name
  without () will return the function object instead of
  the function result.
  There are three ways of writing a function in JavaScript:
  Function Declaration: It declares a function with a

function keyword. The function declaration must have a

function name.

Syntax:

function Hello(paramA, paramB) { // Set of statements

}

Function Expression:

It is similar to a function declaration without the function name. Function expressions can be stored in a variable assignment.

Syntax:

let Hello= function(paramA, paramB) { // Set of statements

}

Example 3: This example explains the usage of the Function expression.

```
const square = function (number) { return number * number;

};
const x = square(4); // x gets the value 16
console.log(x);
```

Output

Functions as Variable Values:

Functions can be used the same way as you use

---

variables.

Example:

```
// Function to convert Fahrenheit to Celsius
function toCelsius(fahrenheit) {

return (fahrenheit - 32) * 5/9; }

// Using the function to convert temperature let
temperatureInFahrenheit = 77;
let temperatureInCelsius =

toCelsius(temperatureInFahrenheit);
let text = "The temperature is " +
temperatureInCelsius + "

Celsius";
```

Arrow Function:

Arrow Function is one of the most used and efficient methods to create a function in JavaScript because of its comparatively easy implementation. It is

comparatively easy implementation. It is a simplified as well as a more compact version of a regular or normal function expression or syntax.

Syntax:

let function_name = (argument1, argument2 ,..) => expression **Example 4:** This example describes the usage of the Arrow function.

```
const a = ["Hydrogen", "Helium", "Lithium", "Beryllium"];

const a2 = a.map(function (s) { return s.length;

});
```

function parameters but haven t discussed them in detail. Parameters are additional information passed to a function. For example, in the above example, the task of the function calcAddition is to calculate the addition of two numbers. These two numbers on which we want to perform the addition operation are passed to this function as parameters. The parameters are passed to the function within parentheses after the function name and separated by commas. A function in JavaScript can have any number of parameters and also at the same time, a function in JavaScript can not have a single parameter.

Example 4: In this example, we pass the argument to the function.

```
function multiply(a, b) {
b = typeof b !== "undefined" ? b : 1; return a * b;

}
```

Calling Functions:

After defining a function, the next step is to call them to make use of the function. We can call a function by using the function name separated by the value of parameters enclosed between the parenthesis and a semicolon at the end. The below syntax shows how to call functions in JavaScript:

Syntax:

functionName(Value1, Value2, ..);

Example 5: Below is a sample program that illustrates the working of functions in JavaScript:

```
function welcomeMsg(name) {
return ("Hello " + name + " welcome to hello");

}

// creating a variable

let nameVal = "Admin"; // calling the function

console.log(welcomeMsg(nameVal));
```

Output:

Hello Admin welcome to hello

Hello Admin welcome to hello

Return Statement:

There are some situations when we want to return some values from a function after performing some operations. In such cases, we can make use of the return statement in JavaScript. This is an optional statement and most of the time the last statement in a JavaScript function. Look at our first example with the function named as calcAddition. This function is calculating two numbers and then returns the result.

Syntax: The most basic syntax for using the return statement is:

```
return value;
```

The return statement begins with the keyword return separated by the value which we want to return from it. We can use an expression also instead of directly returning the value.

How to write a function in

JavaScript ? JavaScript functions serve as reusable blocks of code that can be called from

anywhere within your application. They eliminate the need to repeat the same code, promoting code reusability and modularity. By breaking down a large program into smaller, manageable functions, programmers can enhance code organization and maintainability.

Functions are one of JavaScript's core building elements. In JavaScript, a function is comparable to a procedure—a series of words that performs a task or

calculates a value—but for a process to qualify as a

Function Definition

A **Function Definition** or function statement starts with the function keyword and continues with the following.

- Function's name.

- A list of function arguments contained in
  parenthesis and separated by commas.

- Statements are enclosed in curly brackets.
  Syntax:
  function name(arguments) {
  javascript statements }
  Function Calling:
  Function Calling at a later point in the script, simply type the function's name. By default, all JavaScript functions can utilize argument

objects. Each parameter’s value is stored in an argument object. The argument object is similar to an array. Its values may be accessed using an index, much like an array. It does not, however, provide array methods.

```
function welcome() { console.log("welcome to hello");
}
```

// Function calling

welcome();

Output

welcome to hello

Function Arguments:

A function can contain one or more arguments that are sent by the calling code and can be utilized within the function. Because JavaScript is a dynamically typed programming language, a **Function Arguments** can have any data type as a value.

```
function welcome(name) {
console.log("Hey " + "" + name + " " + "welcome to
hello");

}

// Passing arguments

welcome("Rohan");
```

Output

Hey Rohan welcome to hello

Return Value:

A return statement is an optional part of a JavaScript function. If you wish to return a value from a function,

you must do this. **Return Value** should be the final

Welcome to hello

Function Expression:

We may assign a function to a variable and then utilize that variable as a function in JavaScript. **Function Expression** is known as a function expression.

```
let welcome = function () { return "Welcome to hello";

}
let hello = welcome(); console.log(hello);
```

Output

Welcome to hello

Types Of Functions in Javascript

function several times to give various values to it or run it multiple times.

```
function add(a, b) { return a + b;
}
console.log(add(5, 4));
```

Output

9

2. Anonymous function:

We can define a function in JavaScript without giving it a name. This nameless function is referred to as the **Anonymous function**. A variable must be assigned to an anonymous function.

```
let add = function (a, b) { return a + b;
}
console.log(add(5, 4));
```

Output

9

Functions fall within the purview of the outer function. The inner function has access to the variables and arguments of the outer function. However, variables declared within inner functions cannot be accessed by outer functions.

```
function msg(firstName) { function hey() {
console.log("Hey " + firstName); }
return hey(); }
msg("Ravi");
```

Output

Hey Ravi

4. Immediately invoked function expression:

The browser executes the invoked function expression as soon as it detects it. **Immediately invoked function expression** has the advantage of running instantly where it is situated in the code and producing direct output. That is, it is unaffected by code that occurs later in the script and can be beneficial

beneficial.

```
let msg = (function() {
```

```
return "Welcome to hello" ; })();

console.log(msg);
```

Output

Welcome to hello

JavaScript Function Call

JavaScript function calls involve invoking a function to execute its code. You can call a function by using its name followed by parentheses (), optionally passing arguments inside the parentheses.

Syntax:

call()
Return Value: It calls and returns a method with the owner object being the argument.

JavaScript Function Call Examples **Example 1:** This example demonstrates the use of the call() method

method.

```
// function that returns product of two numbers
function product(a, b) {

return a * b; }

// Calling product() function
let result = product.call(this, 20, 5);

console.log(result);
```

Output

```
100
```

```
//Functions:

//1
function Show(name){ //declaration

console.log("Hi welcome" + " " + name) //Output }
Show("Siri") //Function Calling

//2
function Display(){ //Without parameters

let a=1,b=2;

console.log(a,b) }Display()

//3
function Display1(a1,b1){ //Parameterized functions

console.log(a1,b1) }Display1(4,5)

//4
function Add(a2,b2){ //function without return statement

console.log(a2+b2) }Add(6,9)

//5
```

```
let add1=function(a3,b3){ //function expression
return a3+b3; }
console.log(add1(3,4))
//6
let y=function(){
return "hello"; }
let x=y(); console.log(x);
//7-- Named functions
function Multiply(c,d){ return c*d;
} console.log(Multiply(8,9))
```

```
//8 -- Anonymous function
let multiply1=function(e,f){ return e*f;
}
let result2=multiply1(3,8); console.log(result2)
//9---Nested Function
function First(firstname){ function Second(lastname){
console.log(firstname + " " + lastname) }
Second("GVS"); }First("Bhargavi")
//10--Arrow functions -- lambda functions =>
function Multiply(c,d){ return c*d;
} console.log(Multiply(8,9))
//Arrow function with integers value=(g,h)=>g*h;
console.log(value(6,9))
//Arrow functions with strings
let array=["hello","hi","bye"]
let length=array.map((s)=>s.length) console.log(length)
```

3 Places to put JavaScript code

1. Between the body tag of html
2. Between the head tag of html
3. In .js file (external javaScript)

1) JavaScript Example : code between the body tag

In the above example, we have displayed the dynamic content using JavaScript. Let's see the simple example of JavaScript that displays alert dialog box.

<script type="text/javascript"**>** alert("Hello hello");

</script>

2) JavaScript Example : code between the head tag

Let's see the same example of displaying alert dialog box of JavaScript that is contained inside the head tag.

In this example, we are creating a function msg(). To create function in JavaScript, you need to write function with function_name as given below.

To call function, you need to work on event. Here we are using onclick event to call msg() function.

<html>
<head>

```
<script type="text/javascript"> function msg(){
alert("Hello hello"); }
</script>
</head>
<body>
<p>Welcome to JavaScript</p>
<form>
<input type="button" value="click"
onclick="msg()"/> </form>
</body>
</html>
```

3) External JavaScript file

We can create external JavaScript file and embed it in many html page.

It provides **code re usability** because single JavaScript file can be used in several html pages.

An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.

message.js

```
1 function msg(){
2 alert("Hello hello"); 3}
```

Let's include the JavaScript file into html page. It calls the JavaScript function on button click.

index.html

```
1  <html>

2  <head>

3  <script type="text/javascript"
   src="message.js"></
   script>
```

. **11 </html>**

Advantages of External JavaScript

There will be following benefits if a user creates an external javascript:

. 1 It helps in the reusability of code in more than one HTML file.

. 2 It allows easy code readability.

. 3 It is time-efficient as web browsers cache the external js
files, which further reduces the page loading time.

. 4 It enables both web designers and coders to work with html and js files parallel and separately, i.e., without facing any code conflicting.

. 5 The length of the code reduces as only we need to specify
the location of the js file.

Disadvantages of External JavaScript

There are the following disadvantages of external files:

. 1 The stealer may download the coder's code using the url of the js file.

. 2 If two js files are dependent on one another, then a failure in one file may affect the execution of the other dependent file.

. 3 The web browser needs to make an additional http request to get the js code.

. 4 A tiny to a large change in the js code may cause unexpected results in all its dependent files.

5 We need to check each file that depends on the

commonly created external javascript file.

6 If it is a few lines of code, then better to implement the internal javascript code.

Document Object Model

The **document object** represents the whole

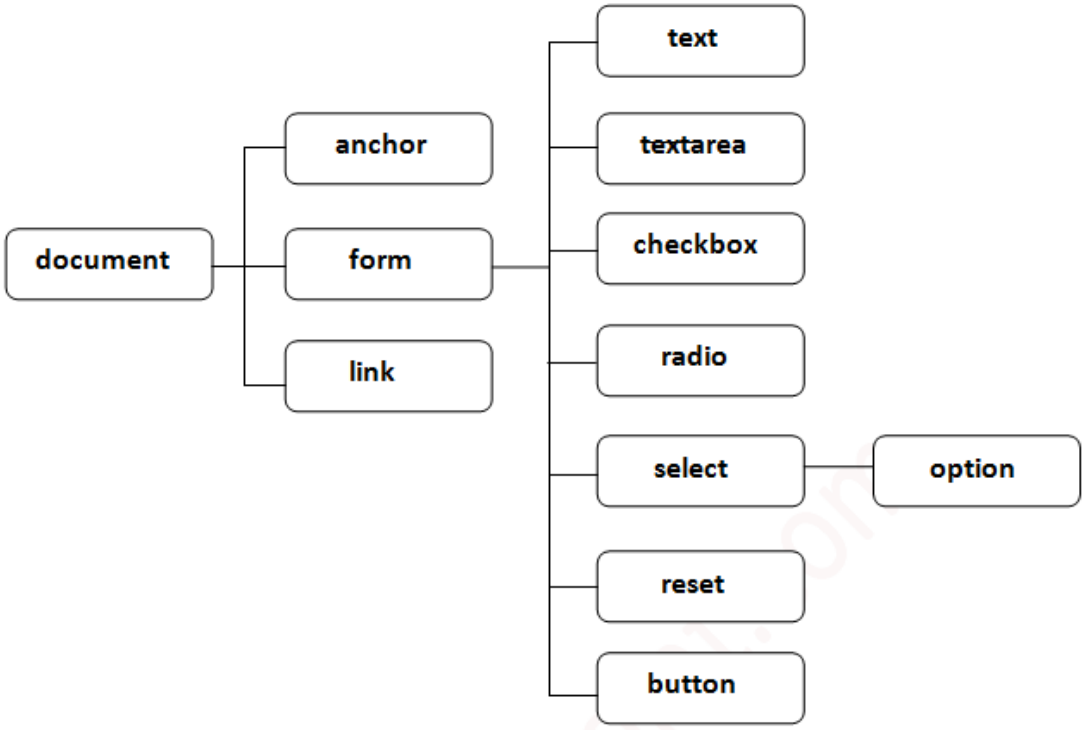
html document.

When html document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

Properties of document object

Let's see the properties of document object that can be

accessed and modified by the document object.



Methods of document object

We can access and change the contents of document by its methods.

The important methods of document object are as follows:

Method

Description

write("string") writes the given string on the

document.

writeln("string") writes the given string on the

document with newline character at the end.

getElementById() returns the element having the

given id value.

getElementsByName() returns all the elements having

the given name value.

getElementsByTagName() returns all the elements having

the given tag name.

getElementsByClassName() returns all the elements having

the given class name.

getElementById() returns the element having the

given id value.

getElementsByName() returns all the elements having

the given name value.

getElementsByTagName() returns all the elements having

the given tag name.

getElementsByClassName() returns all the elements having

the given class name.

Accessing field value by document object

In this example, we are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field.

Here, **document** is the root element that represents the html document.

form1 is the name of the form.

name is the attribute name of the input text.
value is the property, that returns the value of the input text.

Let's see the simple example of document object that prints name with welcome message.

```
<script type="text/javascript">
function printvalue(){
var name=document.form1.name.value;
alert("Welcome: "+name);
}
</script>

<form name="form1">
Enter Name:<input type="text" name="name"/
>
<input type="button" onclick="printvalue()"
value="print name"/
```

>
</form>
Output of the above example

Enter Name:

JavaScript - HTML DOM Methods

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

The HTML DOM document object is the owner of all other objects in your web page.

The HTML DOM Document Object

The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

Finding HTML Elements

Method

document.getElementById(*id*) Find an element by element id

document.getElementsByTagN ame(*name*)

document.getElementsByClass Name(*name*)

Find elements by tag name

Find elements by class name

Changing HTML Elements

D e s c r i p t i o n

Property

element.innerHTML = *new html content*

element.*attribute = new value* Change the attribute value of an

element.style.*property = new style*

Method Description

Change the inner HTML of an element

HTML element

Change the style of an HTML element

D e s c r i p t i o n

i o n

element.innerHTML = *new html content*

element.*attribute = new value* Change the attribute value of an

element.style.*property = new style*

Method Description

element.setAttribute*(attribute, value)*

Change the inner HTML of an element

HTML element

Change the style of an HTML element

Change the attribute value of an HTML element

Adding and Deleting Elements

Method

document.createElement(*eleme nt*)

document.removeChild(*element*)

document.appendChild(*element*)

document.replaceChild(*new, old*)

document.write(*text*) Write into the HTML output

Create an HTML element

Remove an HTML element

Add an HTML element

Replace an HTML element

stream

Adding Events Handlers

D e s c r i p t i o n

Method

document.getElementById(*id*).o nclick = function() {*code*}

Adding event handler code to an onclick event

D e s c r i p t i o n

Finding HTML Elements

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are several ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with id="intro":

Example

```
const element =
document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in element).

If the element is not found, element will contain null.

Finding HTML Elements by Tag Name

This example finds all <p> elements:

Example

```
const element =
document.getElementsByTagName("p");
```

This example finds the element with id="main", and then finds all <p> elements inside "main":

Example

```
const x = document.getElementById("main");
```

const y = x.getElementsByTagName("p");

Finding HTML Elements by Class Name

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

This example returns a list of all elements with class="intro".

Example

```
const x = document.getElementsByClassName("intro");
```

This example returns a list of all <p> elements with class="intro".

Example

```
const x = document.querySelectorAll("p.intro");
```

Finding HTML Elements by HTML Object Collections

This example finds the form element with id="frm1", in the forms collection, and displays all element values:

Example

```
const x = document.forms["frm1"]; let text = "";
for (let i = 0; i < x.length; i++) {

text += x.elements[i].value + "<br>"; }

document.getElementById("demo").innerHTML = text;
```

The HTML DOM allows JavaScript to change the content of HTML elements.

Changing HTML Content

The easiest way to modify the content of an HTML element is by using the innerHTML property.

To change the content of an HTML element, use this syntax:
document.getElementById(*id*).innerHTML = *new HTML*

Changing the Value of an Attribute

To change the value of an HTML attribute, use this syntax:

document.getElementById(*id*).*attribute = new value*
This example changes the value of the src attribute of an <img> element:

Example

<!DOCTYPE html> <html>
<body>

<img id="myImage" src="smiley.gif">

<script>
document.getElementById("myImage").src =
"landscape.jpg"; </script>

</body>
</html>
Example explained:

• The HTML document above contains an <img> element with id="myImage"

- We use the HTML DOM to get the element with id="myImage"

- A JavaScript changes the src attribute of that element from "smiley.gif" to "landscape.jpg"

Dynamic HTML content

JavaScript can create dynamic HTML content:
Date : Fri May 31 2024 11:12:18 GMT+0530 (India Standard Time)

Example

<!DOCTYPE html> <html>
<body>
<script>
document.getElementById("demo").innerHT

ML = "Date : " + Date(); </script>
</body> </html>

document.write()

In JavaScript, document.write() can be used to write directly to the HTML output stream:

Example

```
<!DOCTYPE html> <html>
<body>
<p></p>

<script> document.write(Date()); </script>

<p></p>

</body> </html>
```

JavaScript Forms

JavaScript Form Validation

HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

JavaScript Form

JavaScript Form Validation

JavaScript form validation Example of JavaScript validation JavaScript email validation

It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.

JavaScript provides facility to validate the form on the client- side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form

submit. The user will not be forwarded to the next page until given values are correct.

```
<script>

function validateform(){
var name=document.myform.name.value;
var
password=document.myform.password.value;

if (name==null ll name==""){ alert("Name can't be blank"); return false;

}else if(password.length<6){
alert("Password must be at least 6 characters long."); return false;
}

}

</script>
<body>
<form name="myform" method="post" action="abc.jsp" onsubmit="return validateform()" >
Name: <input type="text" name="name"><br/>
Password: <input type="password" name="password"><br/> <input type="submit" value="register">
</form>
```

JavaScript Retype Password Validation

```
<script type="text/javascript">
function matchpass(){
var
firstpassword=document.f1.password.value;
var
secondpassword=document.f1.password2.valu
e;

if(firstpassword==secondpassword){ return
true;
}
else{

alert("password must be same!"); return false;
}
}

</script>

<form name="f1" action="register.jsp"
onsubmit="return matchpass()">
Password:<input type="password"
name="password" /><br/> Re- enter
Password:<input type="password"
name="password2"/ ><br/>

<input type="submit"> </form>
```

JavaScript Number Validation

Let's validate the textfield for numeric value only. Here, we are using isNaN() function.

```
<script>

function validate(){
var num=document.myform.num.value; if
(isNaN(num)){

document.getElementById("numloc").innerHT
ML="Enter Numeric value only";

return false; }else{

return true;

} }

</script>
<form name="myform" onsubmit="return
validate()" > Number: <input type="text"
name="num"><span id="numloc"></
span><br/>
<input type="submit" value="submit">
</form>
```

JavaScript validation with image

Let’s see an interactive JavaScript form

validation example that displays correct and incorrect image if input is correct or incorrect.

<script>

```
function validate(){
var name=document.f1.name.value;
var password=document.f1.password.value;
var status=false;

if(name.length<1)
{ document.getElementById("nameloc").inner
HTML=
" <img src='unchecked.gif'/> Please enter
your name"; status=false;
```

return status;
}
</script>

<form name="f1" action="#" onsubmit="return validate()"**> <table>**
<tr><td>Enter Name:**</ td><td><input** type="text" name="name"**/>**

<span id="nameloc"**></td></tr>**
<tr><td>Enter Password:**</ td><td><input** type="password" name="password"**/>**
<span id="passwordloc"**></td></tr>**
<tr><td colspan="2"**><input** type="submit" value="register"**/></ td></tr>**
</table>
</form>

JavaScript email validation

We can validate the email by the help of JavaScript.

There are many criteria that need to be follow to validate the

- There must be at least one character before and after the @.

- There must be at least two characters after . (dot).

Let's see the simple example to validate the email field.

```
<script>
function validateemail()
{
var x=document.myform.email.value;
var atposition=x.indexOf("@");
var dotposition=x.lastIndexOf(".");
if (atposition<1 ll
dotposition<atposition+2 ll
dotposition+2>=x.length){
alert("Please enter a valid e-
mail address \n atpostion:"+atposition+"\n
dotposition:"+dotposition);
return false;
} }
</script>
<body>
<form name="myform" method="post"
action="#" onsubmit="return
validateemail();">
Email: <input type="text"
```

name="email"**>
**
<input type="submit" value="register"**> </form>**

JavaScript Events

The change in the state of an object is known as an **Event**. In html, there are various events which represents that some

activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called **Event Handling**. Thus, js handles the HTML events via **Event Handlers**.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are: Mouse events:

Event Performed

Event Handler Description

click onclick When mouse click

mouseover onmouseover When the cursor of

mouseout onmouseout When the cursor of

mousedown onmousedown When the mouse

mouseup onmouseup When the mouse

mousemove onmousemove When the mouse

Keyboard events:

on an element

the mouse comes over the element

the mouse leaves an element

button is pressed over the element

button is released over the element

movement takes place.

Event Performed

Keydown&Keyup onkeydown &

Event Handler Description

onkeyup

When the user press and then release the key

Event Performed

Event Handler Description

Keydown&Keyup onkeydown &

Form events:

onkeyup

submit onsubmit When the user

blur onblur When the focus is

change onchange When the user

When the user press and then release the key

Event Performed

Event Handler Description

focus onfocus When the user

focuses on an element

submits the form

away from a form element

modifies or changes the value of a form element

Window/Document events

Event Performed

Event Handler Description

load onload When the browser

unload onunload When the visitor

resize onresize When the visitor

finishes the loading of the page

leaves the current webpage, the browser unloads it

resizes the window of the browser

Click Event

```
<html>
<head> Javascript Events </head>
<body>
<script language="Javascript" type="text/
Javascript">

<!--
function clickevent() {

document.write("This is hello"); }

//--> </script>

<form>
<input type="button" onclick="clickevent()"
value="Who's this?"/>
</form>
</body>
</html>
```

MouseOver Event

```
<html>
<head>
<h1> Javascript Events </h1>
</head>
<body>
<script language="Javascript" type="text/
Javascript">

<!--
```

function mouseoverevent() {

alert("This is hello"); }

//--> **</script>**

<p onmouseover="mouseoverevent()"**>** Keep cursor over me**</ p>**
</body>
</html>

Focus Event

<html>
<head> Javascript Events**</head>**
<body>
<h2> Enter something here**</h2>**
<input type="text" id="input1" onfocus="focusevent()"**/> <script>**
<!--

function focusevent() {

document.getElementById("input1").style.background=" aqua";

} //**-->**

</script> </body>

</html> Keydown Event

<html>
<head> Javascript Events**</head>**

<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1"
onkeydown="keydownevent()"/ >
<script>
<!--

function keydownevent() {
document.getElementById("input1");

alert("Pressed a key"); }

//--> </script> </body> </html>

Load event

<html>
<head>Javascript Events</head>
</br>
<body onload="window.alert('Page
successfully loaded');"> <script>
<!--
document.write("The page is loaded
successfully");
//-->
</script>
</body>
</html>

JavaScript

addEventListener()

The **addEventListener()** method is used to attach an event handler to a particular element. It does not override the existing event handlers. Events are said to be an essential part of the JavaScript. A web page responds according to the event that occurred. Events can be user-generated or generated by API's. An event listener is a JavaScript's procedure that waits for the occurrence of an event.

The addEventListener() method is an inbuilt function of JavaScript. We can add multiple event handlers to a particular

defined as a string that specifies the event's name.

Note: Do not use any prefix such as "on" with the parameter value. For example, Use "click" instead of using "onclick".

function: It is also a required parameter. It is a JavaScript function which responds to the event occur.

useCapture: It is an optional parameter. It is a Boolean type value that specifies whether the event is executed in the bubbling or capturing phase. Its possible values are **true** and **false**. When it is set to true, the event handler executes in the capturing phase. When it is set to false, the handler executes in the bubbling phase. Its default value is **false**.

Let's see some of the illustrations of using the addEventListener() method.

Example

It is a simple example of using the addEventListener() method. We have to click the given HTML button to see the effect.

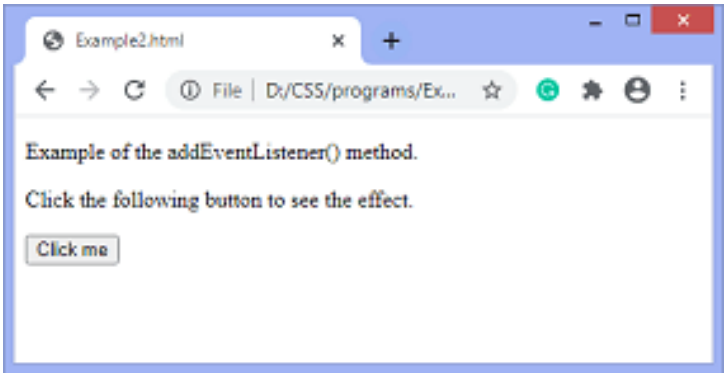
** ** Example of the addEventListener() method. ** ** Click the following button to see the effect. ** <button** id = "btn"**>** Click me **</button>**

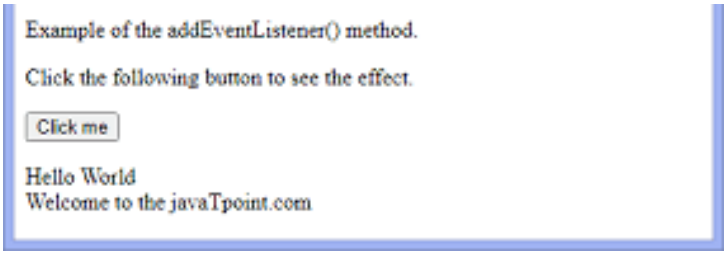
<p id = "para"**> **
<script>
document.getElementById("btn").addEventListener("click", fun);

function fun() {
document.getElementById("para").innerHTML = "Hello World" + "**
**" + "Welcome to the hello.com";
}
</script>
</body>
</html>

Output

After clicking the given HTML button, the output will be -





Now, in the next example we will see how to add many events to the same element without overwriting the existing events.

Example

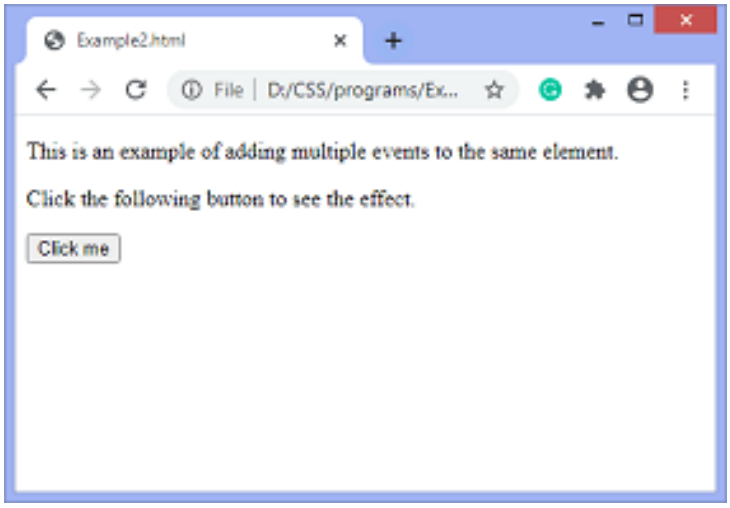
In this example, we are adding multiple events to the same element.

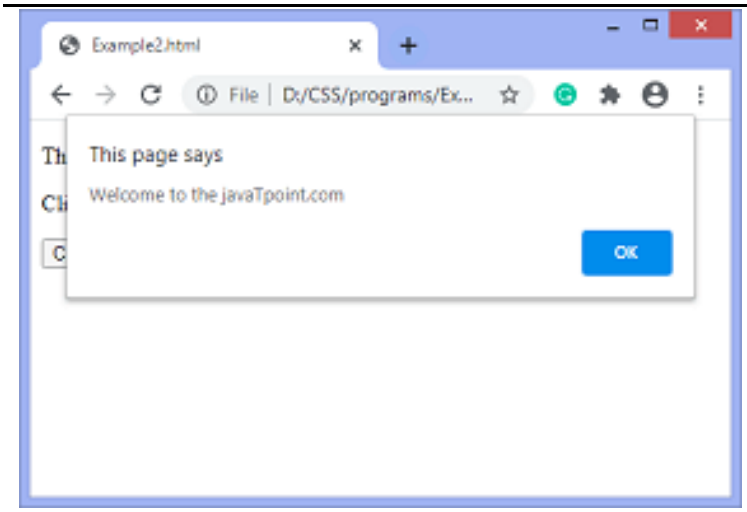
```
<!DOCTYPE html>
<html>
<body>
<p> This is an example of adding multiple events to the same element. </p>

<p> Click the following button to see the effect. </p> <button id = "btn"> Click me </button>
<p id = "para"></p>
<p id = "para1"></p>

<script>

function fun() {
alert("Welcome to the hello.com");

}

function fun1()
```

```
function fun1()
{ document.getElementById("para").innerHTML = "This is

second function"; }
function fun2()
{ document.getElementById("para1").innerHTML = "This is

third function";
}
var mybtn = document.getElementById("btn");
mybtn.addEventListener("click", fun);
mybtn.addEventListener("click", fun1);
mybtn.addEventListener("click", fun2); </script>
</body>
</html>
```

Output

Now, when we click the button, an alert will be displayed. After clicking the given HTML button, the output will be -





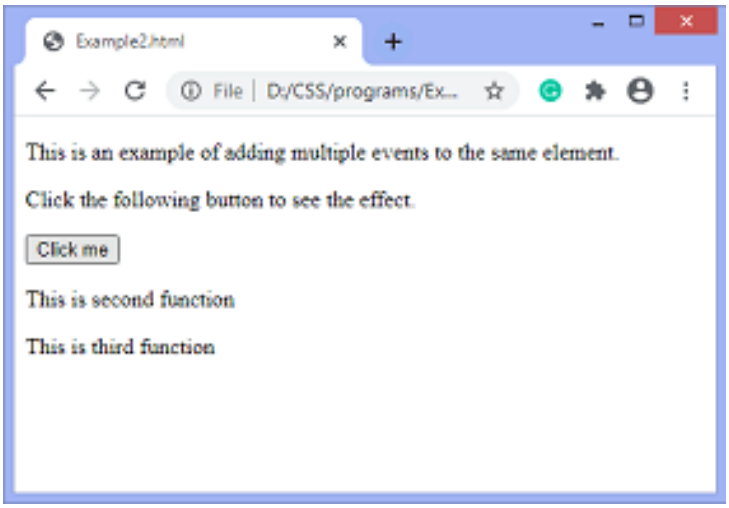
When we exit the alert, the output is -

Example

In this example, we are adding multiple events of a different type to the same element.

```
<!DOCTYPE html>
<html>
<body>
<p> This is an example of adding multiple events of different type to the same element.
</p>

<p> Click the following button to see the
```

effect. **</p>**



<button id = "btn"**>** Click me **</button> <p** id = "para"**></p>**
<script>
function fun() {

btn.style.width = "50px"; btn.style.height = "50px"; btn.style.background = "yellow"; btn.style.color = "blue";

}

function fun1()
{ document.getElementById("para").innerHTML = "This is

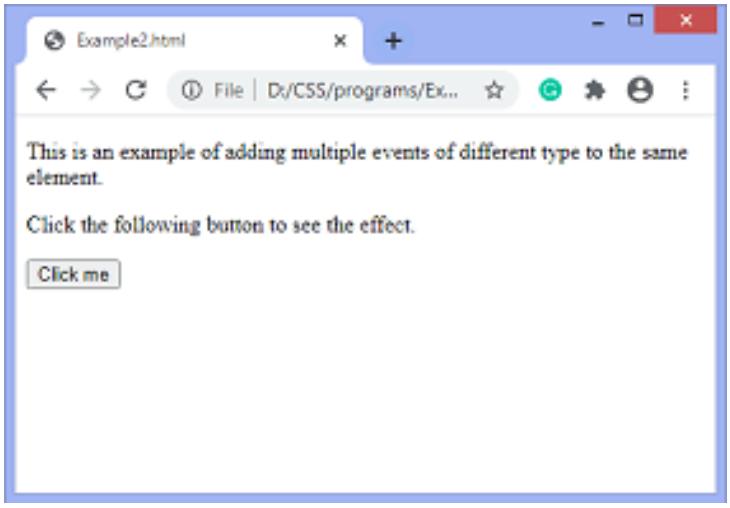
second function";

}
function fun2() {

btn.style.width = ""; btn.style.height = "";

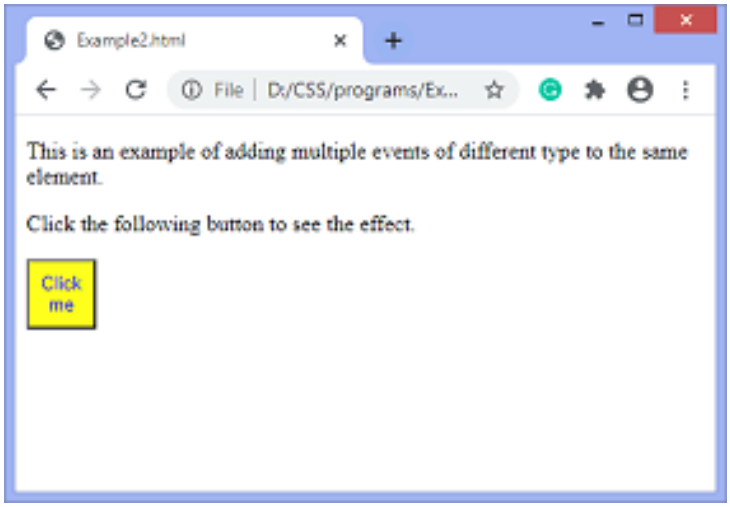
```
btn.style.background = ""; btn.style.color = "";

}
var mybtn = document.getElementById("btn");
mybtn.addEventListener("mouseover", fun);
mybtn.addEventListener("click", fun1);
mybtn.addEventListener("mouseout", fun2); </
script>
</body>
</html>
```

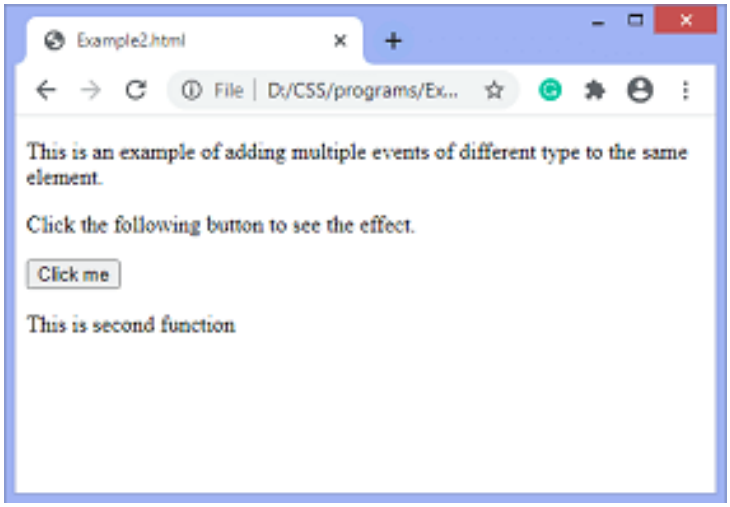
Output



When we move the cursor over the button, the output will be -

After clicking the button and leave the cursor, the output will be -





Event Bubbling or Event Capturing

Now, we understand the use of the third parameter of

JavaScript's addEventListener(), i.e., ***useCapture.***

In HTML DOM, **Bubbling** and **Capturing** are the two ways of event propagation. We can understand these ways by taking an example.

Suppose we have a div element and a

paragraph element inside it, and we are applying the **"click"** event to both of them using the **addEventListener()** method. Now the question is on clicking the paragraph element, which element's click event is handled first.

So, in **Bubbling,** the event of paragraph element is handled first, and then the div element's event is handled. It means that in bubbling, the inner element's event is handled first, and then the outermost element's event will be handled.

In **Capturing** the event of div element is handled first, and then the paragraph element's event is handled. It means that in capturing the outer element's event is handled first, and then the innermost element's event will be handled.

Example

In this example, there are two div elements. We can see the bubbling effect on the first div element and the capturing effect on the second div element.

When we double click the span element of the first div element, then the span element's event is handled first than the div element. It is called *bubbling*.

But when we double click the span element of the second div element, then the div element's event is handled first than the span element. It is called *capturing*.

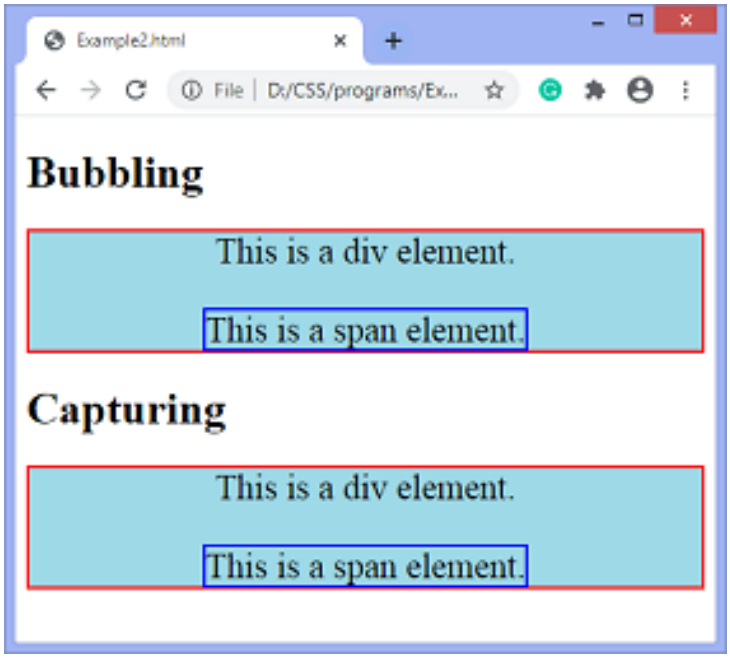
<!DOCTYPE html**> <html>**
<head>
<style>

div{
background-color: lightblue; border: 2px solid red; font-size: 25px;
text-align: center;
}
span{
border: 2px solid blue;
}
</style>

```
</head>
<body>
<h1> Bubbling </h1>
<div id = "d1">
This is a div element.
<br><br>
<span id = "s1"> This is a span element. </
span> </div>
<h1> Capturing </h1>
<div id = "d2"> This is a div element.
<br><br>
<span id = "s2"> This is a span element. </
span> </div>

<script>

document.getElementById("d1").addEventListe
ner("dblclick", function() {alert('You have double
clicked on div element')}, false);
document.getElementById("s1").addEventListe
ner("dblclick", function() {alert('You have double
clicked on span element')}, false);
document.getElementById("d2").addEventListe
ner("dblclick", function() {alert('You have double
clicked on div element')}, true);
document.getElementById("s2").addEventListe
ner("dblclick", function() {alert('You have double
clicked on span element')}, true);
```

</script> </body> </html>

Output



We have to double click the specific elements to see the effect.

Introduction to ES6

ES6 or ECMAScript 2015 is the 6th version of the ECMAScript programming language. ECMAScript is the standardization of Javascript which was released in 2015 and subsequently renamed as ECMAScript 2015. ECMAScript and Javascript are both different.

ECMAScript vs Javascript

ECMAScript: It is the specification defined in ECMA-262 for creating a general-purpose scripting language. In simple terms, it is a standardization for

creating a scripting language. It was introduced by ECMA International and is an implementation with which we learn how to create a scripting language.

Javascript: A general-purpose scripting language that confirms the ECMAScript specification. It is an implementation that tells us how to use a scripting language.

ES6

Javascript ES6 has been around for a few years now, and it allows us to write code in a clever way which basically makes the code more modern and more readable. It's fair to say that with the use of ES6 features we write less and do more, hence the term 'write less, do more' definitely suits ES6.

Table of Content

- Promises
- Symbol
- String Methods
- Array Methods
- Object Enteries

const

The **const** keyword is used to declare constant variables whose values can't be changed. It is an **immutable** variable except when used with objects.

Example: The below example will explain you the use of **const** keyword in different situations.

```
// Const
const name = 'Mukul';
console.log(name);
// Will print 'Mukul' to the console.

// Trying to reassign a const variable
name = 'Rahul';
console.log(name);
// Will give TypeError.

// Trying to declare const variable first
// and then initialise in another line
const org_name;
org_name = "hello";
console.log(org_name);
// Throws an syntax error: missing initializer in const declaration
```

let

The **let** variables are mutable i.e. their values can be changed. It works similar to the var keyword with some

key differences like scoping which makes it a better option when compared to var.

Example: This example will illustrate how to declare variable using the let keyword.

```
// let
let name = 'Mukul';
console.log(name);
// Prints Mukul

name = 'Rahul';
console.log(name);
// Prints Rahul

// Trying to declare let variable first and then initialise in another line
let org_name;
org_name = "hello";
console.log(org_name);
// Prints hello
```

Output

Mukul
Rahul
hello

Arrow functions

Arrow functions are a more concise syntax for writing function expressions. These function expressions make your code more readable, and more modern.

Example: The below example will show you how the arrow functions can be created and implemented.

```
// Function declaration using function keyword
function simpleFunc(){
    console.log("Declared using the function keyword");
}
simpleFunc();

// Function declared using arrow functions
const arrowFunc = () => {
    console.log("Declared using the arrow functions");
}
arrowFunc();
```

Output

Declared using the function keyword
Declared using the arrow functions

Template literal

It allows us to use the JavaScript variables with the string without using the **'+'** operator. Template literal defined using **(")** quotes.

Example: This example will explain the practical use of template literals.

```
// Without Template Literal
var name = 'hello';
console.log('Printed without using Template Literal');
console.log('My name is '+ name);

// With Template Literal
const name1 = 'hello';
console.log('Printed by using Template Literal');
console.log(`My name is ${name1}`);
```

Output

Printed without using Template Literal
My name is hello
Printed by using Template Literal
My name is hello

Object and Array Destructuring

Destructing in javascript basically means the breaking down of a complex structure(Objects or arrays) into simpler parts. With the destructing assignment, we can ‘unpack’ array objects into a bunch of variables.

Example: The below example will explain how the destructuring can be done in ES6.

```
// Object Destructuring
const college = {
    name : 'DTU',
    est : '1941',
    isPvt : false
};
let {name, est, isPvt} = college;
console.log(name, est, isPvt);

// Array Destructuring
const arr = ['lionel', 'messi', 'barcelona'];
let[value1,value2,value3] = arr;
console.log(value1, value2, value3);
```

Output

DTU 1941 false
lionel messi barcelona

Default Parameters

In ES6, we can declare a function with a default parameter with a default value.

Example: This example will show how to define default parameters for a function.

```
function fun(a, b=1){
    return a + b;
}
console.log(fun(5,2));
console.log(fun(3));
```

Output

```
7
4
```

Classes

ES6 introduced classes in JavaScript. Classes in javascript can be used to create new Objects with the help of a constructor, each class can only have one constructor inside it.

Example: The below example will show how to create classes in ES6.

```
// classes in ES6
class Vehicle{
    constructor(name,engine){
        this.name = name;
        this.engine = engine;
    }
}

const bike1 = new Vehicle('Ninja ZX-10R','998cc');
const bike2 = new Vehicle('Duke','390cc');

console.log(bike1.name); // Ninja ZX-10R
console.log(bike2.name); // Duke
```

Output

```
Ninja ZX-10R
```

Duke

Rest parameter and spread operator

Rest Parameter: It is used to represent a number of parameter in an array to pass them together to a function.

Example: This example will illustrate the practical use of the Rest paramter.

```
// ES6 rest parameter
function fun(...input){
    let sum = 0;
    for(let i of input){
        sum += i;
    }
    return sum;
}
console.log(fun(1,2)); // 3
console.log(fun(1,2,3)); // 6
console.log(fun(1,2,3,4,5)); // 15
```

Output

```
3
6
15
```

Spread Operator: It simply changes an array of **n** elements into a **list of n different elements**.

Example: The below example will explain the practical implementation of the Spread Operator.

```
// Spread operator
let arr = [1,2,3,-1];
console.log(...arr);
console.log(Math.max(...arr)); // -1
```

Output

```
1 2 3 -1
3
```

for/of Loop

The for/of loop allows you to iterate through the iterable items but in a short syntax as compared to other loops.

Example: The for/of loop is implemented in the below code example with an array

```
const myArr = [5, 55, 33, 9, 6]
for(let element of myArr){
    console.log(element);
}
```

Output

```
5
55
33
9
6
```

JavaScript Maps **and** Sets

Map: The **maps** in JavaScript are used to store multiple items in the form of key-value pairs. The keys of a map can be of any datatype.

Set: A **set** consist of only unique value, a value can be stored only once in a set. It can store any value of any

datatype.

Example: The below example will explain the use of both the JavaScript Map and Set practically.

```
// Maps in JavaScript
const myMap = new Map([
    ["stringKey", 23],
    [48, "numberedKey"]
]);
myMap.set(false, 0);
myMap.set(1, true);
console.log(myMap.get("stringKey"),
    myMap.get(48), myMap.get(false),
    myMap.get(1));

// Sets in JavaScript
const mySet =
    new Set(["string value", "string value"]);
mySet.add(24);
mySet.add(24);
mySet.add(3);
console.log(mySet);
```

Output

```
23 numberedKey 0 true
Set(3) { 'string value', 24, 3 }
```

Promises

JavaScript promises are used to perform the asynchronous tasks, that takes some time to get executed. It combines the synchronous and asynchronous code together.

Example: The below example will show how you can create and implement a promise in JavaScript.

```
const JSpromise = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve("Promise is Working")
    }, 2000);
});

JSpromise.then((value) => {console.log(value)});
```

Output

Promise is Working

Symbol

Symbol is a type of primitive data type intoduced in ES6. It is used to specify the **hidden** identifiers that can not be directly accessed by any other code.

Example: The use of the Symbol data type is explained in the below code example.

```
const hello = {
    name: "hello",
    desc: "A Computer Science portal for all hellos."
}

let short_name = Symbol("short_name")
hello.short_name = "hello";
console.log(`${hello.name}, \n${hello.desc}`);
console.log(`Company's Short Name using hello.short_name: $
{hello.short_name} `)
console.log(`Company's Short Name using hello[short_name]: $
{hello[short_name]} `)
```

Output

hello,
A Computer Science portal for all hellos.

Company's Short Name using hello.short_name: hello
Company's Short Name using hello[short_name]: undefined

String Methods

- **JavaScript startsWith():** This method will return **true** only if the testing string starts with the passed or specified string.
- **JavaScript endsWith():** This mthod will return **true,** if the string ends with the passes or specified string value.
- **JavaScript includes():** It will return **true,** if the testing string contains the specified or passes value.

Example: The below code example will illustrate the use of all the string methods introduced in ES6.

```
// String startsWith()
const useStart = "This string implements the startsWith() method.";
console.log(useStart.startsWith("This string"),
useStart.startsWith("This is"));

// String endsWith()
const useEnd = "This string implements the endsWith() method.";
console.log(useEnd.endsWith("clear() method."),
useEnd.endsWith("method."));

// String includes()
const useIncludes = "This string implements the includes() method.";
console.log(useIncludes.includes("includes()"),
useIncludes.includes("My name"));
```

Output

```
true false
false true
true false
```

Array Methods

- **JavaScript Array.from():** It will return a array from any object which is iterable and has the length property associated with it.
- **JavaScript Array.keys():** It returns an array of the iterator keys of the array.
- **JavaScript Array.find():** It will return the value of the first array element that matches or passes the condition of the passed function.
- **JavaScript Array.findIndex():** It will return the index of the first array element that matches or passes the condition of the passed function.

Example: The below code example implements all array methods introduced in ES6.

```
// Array.from() method
const newArr = Array.from("hellodforhellos");
console.log("Implementing Array.from(): ", newArr)

// Array.keys() method
const milkProducts = ["Curd", "Cheese", "Butter", "Ice-Cream"];
const arrayKeys = milkProducts.keys();
console.log("Implementing Array.keys(): ")
for(let key of arrayKeys){
    console.log(key)
}

// Array.find() method
const findArray = ["clock", "strong", "planet", "earth"];
const lessThanSix = (item) => {
    return item.trim.length < 6;
}
console.log("Implementing Array.find(): ",
findArray.find(lessThanSix));
console.log("Implementing Array.findIndex(): ",
findArray.findIndex(lessThanSix));
```

Output

```
Implementing Array.from():  [
  'G', 'e', 'e', 'k',
  'd', 'f', 'o', 'r',
  'G', 'e', 'e', 'k',
  's'
]
Implementing Array.keys():
0
1
2
3
Implementing Array.find():  clock
Implementing Array.findInd...
```

Object Entries

Object.entries() method is used to convert a single

valued array into an array object with a **key-value** pair as array items.

Example: The below example implements the object.entries() method practically.

```
const myArr =
   ["hello", "A Computer Science Portal for all hellos"];

const arr = myArr.entries()
for(let item of arr){
   console.log(item);
}
```

Output

```
[ 0, 'hello' ]
[ 1, 'A Computer Science Portal for all hellos' ]
```

Error Handling:

JavaScript Errors Throw and Try to Catch

In JavaScript, errors can be thrown using the throw statement to indicate an exceptional condition. The try block is used to wrap code that might throw an error, and the catch block handles the error, preventing the program from crashing and allowing graceful error management.
But all errors can be solved and to do so we use five statements that will now be explained.

- The **try** statement lets you test a block of code to check for errors.
- The **catch** statement lets you handle the error if any

are present.

- The **throw** statement lets you make your own errors.
- The **finally** statement lets you execute code after try and catch.
  The **finally** block runs regardless of the result of the try-catch block.

Example:

```
try {
    dadalert("Welcome Fellow hello!");
}
catch (err) {
    console.log(err);
}
```

Output: In the above code, we make use of ‘dadalert’ which is not a reserved keyword and is neither defined hence we get the error.

```
function helloFunc() {
    let a = 10;
    try {
        console.log("Value of variable a is : " + a);
    }
    catch (e) {
        console.log("Error: " + e.description);
```

```
    }
}
helloFunc();
```

Output: In the above code, our catch block will not run as there's no error in the above code and hence we get the output 'Value of variable a is: 10'.

Try and Catch Block

The **try** statement allows you to check whether a specific block of code contains an error or not. The **catch** statement allows you to display the error if any are found in the try block.

```
try {
    Try Block to check for errors.
}
catch(err) {
     Catch Block to display errors.
}
```

Javascript Throws Block The `throw` Statement

When any error occurs, JavaScript will stop and generate an error message. The throw statement lets you create your own custom error. Technically you can throw your custom exception (throw an error). The exception can be a JavaScript Number, String, Boolean, or Object. By using throw together with try and catch, you can easily control the program flow and generate custom error messages.

```
try {
    throw new Error('Yeah... Sorry');
}
catch (e) {
    console.log(e);
}
```

The finally Block

The finally Statement runs unconditionally after the execution of the try/catch block. Its syntax is :

```
try {
     Try Block to check for errors.
}
catch(err) {
      Catch Block to display errors.
}
finally {
      Finally Block executes regardless of the try / catch result.
}
try {
    console.log('try');
} catch (e) {
```

```
    console.log('catch');
} finally {
    console.log('finally');
}
```

Output: The Finally Block can also override the message of the catch block so be careful while using it.

- **JSON**
    - JSON is a format for storing and transporting data.
    - JSON is often used when data is sent from a server to a web page.
    - **What is JSON?**
    -

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight data interchange format
- JSON is language independent *
- JSON is "self-describing" and easy to understand
    -

    - The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text only. Code for reading and generating JSON data can be written in any programming language.

The JSON Format Evaluates to JavaScript Objects:

- The JSON format is syntactically identical to the code for creating JavaScript objects.

- Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.

JSON Syntax Rules:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

JSON Data - A Name and a Value

JSON data is written as name/value pairs, just like JavaScript object properties.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

"firstName":"John"
JSON names require double quotes. JavaScript names do not.

JSON Objects

JSON objects are written inside curly braces.

Just like in JavaScript, objects can contain multiple name/value pairs:

{"firstName":"John", "lastName":"Doe"}

JSON Arrays

JSON arrays are written inside square brackets.

Just like in JavaScript, an array can contain objects:

```
"employees":[
  {"firstName":"John", "lastName":"Doe"},
```

```
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter", "lastName":"Jones"}
]
```

In the example above, the object "employees" is an array. It contains three objects.

Each object is a record of a person (with a first name and a last name).

Converting a JSON Text to a JavaScript Object:

A common use of JSON is to read data from a web server, and display the data in a web page.

For simplicity, this can be demonstrated using a string as input.

First, create a JavaScript string containing JSON syntax:

```
let text = '{ "employees" : [' +
'{ "firstName":"John" , "lastName":"Doe" },' +
'{ "firstName":"Anna" , "lastName":"Smith" },' +
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';
```

Then, use the JavaScript built-in function JSON.parse() to convert the string into a JavaScript object:

```
const obj = JSON.parse(text);
```

Finally, use the new JavaScript object in your page.

A common use of JSON is to exchange data to/from a web server.

When sending data to a web server, the data has to be a string.

Convert a JavaScript object into a string with JSON.stringify().

Stringify a JavaScript Object:

Imagine we have this object in JavaScript:

const obj = {name: "John", age: 30, city: "New York"};
Use the JavaScript function JSON.stringify() to convert it into a string.

const myJSON = JSON.stringify(obj);
The result will be a string following the JSON notation.

myJSON is now a string, and ready to be sent to a server.

Stringify a JavaScript Array:

It is also possible to stringify JavaScript arrays:

Imagine we have this array in JavaScript:

const arr = ["John", "Peter", "Sally", "Jane"];
Use the JavaScript function JSON.stringify() to convert it into a string.

const myJSON = JSON.stringify(arr);

Difference Between Local Storage, Session Storage And Cookies:

The HTTP protocol is one of the most important protocols for smooth communication between the server and the client. The main disadvantage of the HTTP protocol is that it is a stateless protocol, which means it does not track any kind of response or request by the server or the client. So in order to resolve this problem, there are three ways to track useful information. In this article, we are going to see the difference between local storage, session storage, and cookies and why it’s important for a web developer to know these terms.

Local Storage: This read-only interface property provides access to the document's local storage object; the stored data is stored across browser sessions. Similar to sessionStorage, except that sessionStorage data gets cleared when the page session ends—that is, when the page is closed. It is cleared when the last "private" tab of a browser is closed (localStorage data for a document loaded in a private browsing or incognito session).

DOMStrings are storage formats that use UTF-16 to encode data, which uses two bytes per character. Strings are automatically generated from integer keys, just as they are for objects. The data stored in LocalStorage is specific to a protocol in the document. If the site is loaded over HTTP (e.g., http://example.com), localStorage returns a different object than if it is loaded over HTTPS (e.g., https://abc.com).

If a document is loaded from a file URL (that is, directly from the user's local file system instead of being loaded from the server), the requirements for behavior are undefined and may vary among different browsers. Each file appears to be returned a different object by localStorage in all current browsers: URL. Essentially, it seems to be the case that each URL file has its own unique local storage area.

This behavior cannot be guaranteed because, as mentioned above, the file URL requirements remain unclear. As such, there's a possibility that browsers may change how they handle files at any time. The way

some browsers handle it has evolved.

Local storage has 4 methods:

- **setItem() Method –** This method takes two parameters one is key and another one is value. It is used to store the value in a particular location with the name of the key.

localStorage.setItem(key, value)

- **getItem() Method –** This method takes one parameter that is key which is used to get the value stored with a particular key name.

localStorage.getItem(key)

- **removeItem() Method –** This is method is used to remove the value stored in the memory in reference to key.

localStorage.removeItem(key)

- **clear() Method –** This method is used to clear all the values stored in localstorage.

localStorage.clear()

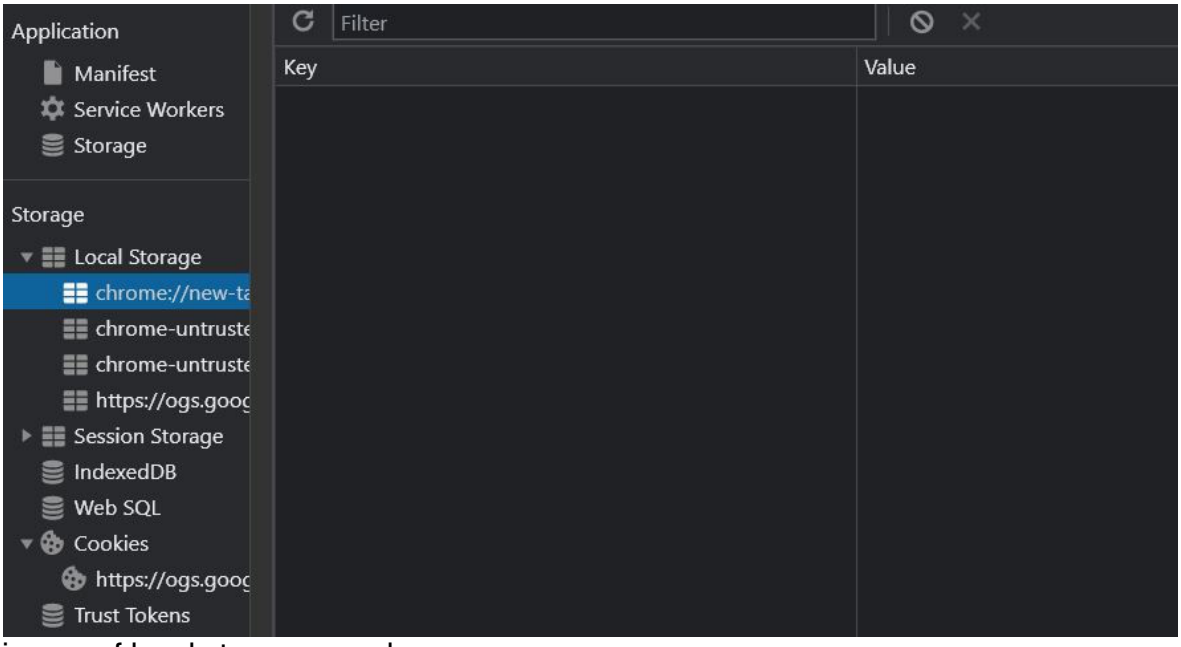


image of local storage panel

What is Session Storage?

Session Storage objects can be accessed using the sessionStorage read-only property. The difference between sessionStorage and localStorage is that localStorage data does not expire, whereas sessionStorage data is cleared when the page session ends.

A unique page session gets created once a document is loaded in a browser tab. Page sessions are valid for only one tab at a time. Pages are only saved for the amount of time that the tab or the browser is open; they do not persist after the page reloads and restores. A new session is created each time a tab or window is opened; this is different from session cookies. Each tab/window that is opened with the same URL creates its own sessionStorage.When you duplicate a tab, the sessionStorage from the original tab is copied to the duplicated tab. Closing a window/tab ends the session and clears sessionStorage objects.

A page's protocol determines what data is stored in sessionStorage. Particularly, data stored by scripts accessed through HTTP (for example, http://abc.com) is stored in a separate object from the same site accessed through HTTPS (for instance, https://abc.com). A DOMString number is two bytes per character in UTF-16 DOMString format. Strings are automatically generated from integer keys just as they are for objects.

Session Storage has 4 methods:

- **setItem() Method –** This method takes two

parameters one is key and another one is value. It is used to store the value in a particular location with the name of the key.

sessionStorage.setItem(key, value)

- **getIteam() Method –** This method takes one parameter that is key which is used to get the value stored with a particular key name.

sessionStorage.getItem(key)

- **removeItem() Method –** This is method is used to remove the value stored in the memory in reference to key.

sessionStorage.removeItem(key)

- **clear() Method –** This method is used to clear all the values stored in the session storage

sessionStorage.clear()

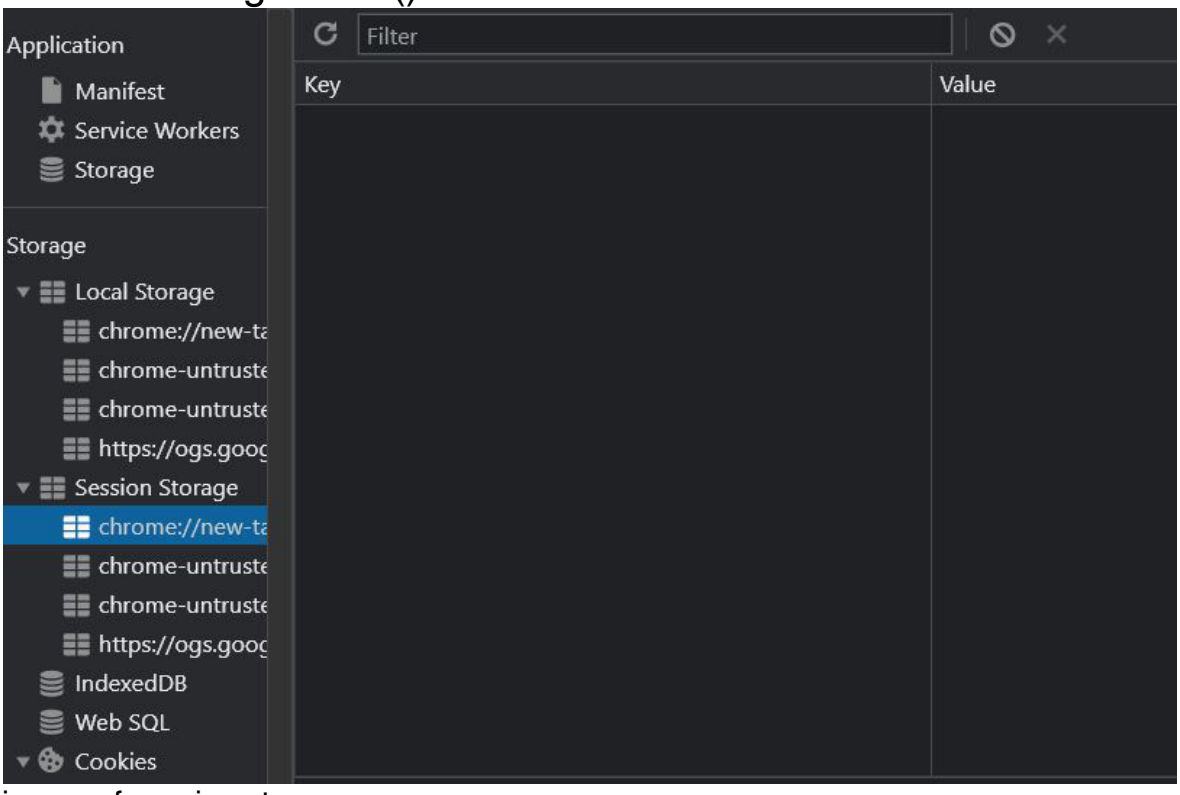


image of session storage

Cookie: The term “cookie” refers to just the textual information about a website. In order to recognize you and show you results according to your preferences, this

website saves some information in your local system when you visit a particular website. The history of the internet has long been marked by the use of cookies. A website visitor asks the server for a web page when they visit it. Every request for a server is unique. Likewise, if you visit a hundred times, each request will be considered unique by the server. Since a server receives many requests every second, storing every user's information on a server doesn't seem logical and obvious. The same information may not be needed again if you don't return. Therefore, a cookie is sent and stored on your local machine to uniquely identify you. You will receive a response from the same server the next time you hit it since it recognizes you. Almost every server uses this cookie (some exceptions exist today because of advertisements). Therefore, although you might have many cookies in your system, such cookies will be recognized by a server and analyzed.

When cookies were first developed, they were used to better the developer's experience. Consider visiting a website in a language other than your native one (let's say English). You can select English as your language from the website's language section. It might be necessary to switch languages five times a day if you visit the same website five times. These details are therefore stored in a cookie on your system. This ensures that the server knows that you wish to view the website in English the next time you send a request. Cookies are vital in this regard. The scale cookies used today are much smaller than the example above.

LocalStorage, SessionStorage:

Web storage objects localStorage and sessionStorage allow to save key/value pairs in the browser.

What's interesting about them is that the data survives a page refresh (for sessionStorage) and even a full browser restart (for localStorage).

We already have cookies. Why additional objects?

> Unlike cookies, web storage objects are not sent to server with each request. Because of that, we can store much more. Most modern browsers allow at least 5 megabytes of data (or more) and have settings to configure that.
>
> Also unlike cookies, the server can't manipulate storage objects via HTTP headers. Everything's done in JavaScript.
>
> The storage is bound to the origin (domain/protocol/port triplet). That is, different protocols or subdomains infer different storage objects, they can't access data from each other.

Both storage objects provide the same methods and properties:

> setItem(key, value) – store key/value pair.
>
> getItem(key) – get the value by key.
>
> removeItem(key) – remove the key with its value.
>
> clear() – delete everything.
>
> key(index) – get the key on a given position.
>
> length – the number of stored items.

As you can see, it's like a Map collection (setItem/getItem/removeItem), but also allows access by index with key(index).

Working with APIs:

What is Web API?

- API stands for **A**pplication **P**rogramming **I**nterface.

- A Web API is an application programming interface for the Web.

- A Browser API can extend the functionality of a web browser.

- A Server API can extend the functionality of a web server.

Browser APIs

- All browsers have a set of built-in Web APIs to support complex operations, and to help accessing data.

Third Party APIs

- Third party APIs are not built into your browser.

- To use these APIs, you will have to download the code from the Web.

Examples:

- YouTube API - Allows you to display videos on a web site.
- Twitter API - Allows you to display Tweets on a web site.
- Facebook API - Allows you to display Facebook info on a web site.

How to connect to an API in JavaScript ?

An **API** or Application Programming Interface is an intermediary which carries request/response data between the **endpoints** of a channel. We can visualize an analogy of API to that of a waiter in a restaurant. A typical waiter in a restaurant would welcome you and ask for your order. He/She confirms the same and

carries this message and posts it to the order queue/ kitchen. As soon as your meal is ready, he/she also retrieves it from the kitchen to your table. As such, in a typical scenario, when a client endpoint requests a resource from a resource server endpoint, this request is carried via the API to the server. There are a various relevant information that are carried by the APIs which conform to certain specification of schema such as that provided by **OpenAPI**, **GraphQL** etc. This information may include endpoints URL, operations(GET, POST, PUT etc), authentication methods, tokens, license and other operational parameters. APIs most commonly follow the **JSON** and **XML** format as its key mode of exchanging request/response while some also adhere to the **YAML**format.

Mostly the response generated by a resource server is a JSON schema which carries the state information of the resource requested at the other endpoint. As such these APIs have been named **REST APIs** where **REST** stands for **RE**presentational **S**tate **T**ransfer. The state of the resources can be affected by the API operations. It should also be noted that there are System APIs as well which are used by the operating system to access kernel functions. A common example would include the **Win32** API which is a windows platform API and acts as a bridge for system level operations such as File/Folder select, button styling etc. Most programming languages which have GUI libraries are wrapped upon this layer.

Example:


```
{
   "location": {
     "lat": 41.1,
     "lng": -0.1
   },
   "accuracy": 1200.2
 }
```


How to use JavaScript Fetch API to Get Data?
An API (Application Programming Interface) is a set of rules, protocols, and tools that allows different software applications to communicate with each other.

One of the popular ways to perform API requests in JavaScript is by using **Fetch API**. Fetch API can make **GET** and **POST** requests, JavaScript's Fetch API is a powerful tool for developers seeking to retrieve this data efficiently. This guide focuses on using Fetch API to master the art of "**GET**" requests, the essential method for gathering information from APIs.

What is the JavaScript Fetch API?

The **Fetch API** provides an interface for fetching resources (like JSON data) across the web. It offers a more powerful and flexible alternative to traditional XMLHttpRequest.

How to Use the JavaScript Fetch API

JavaScript fetch API uses the fetch() function at its core. The fetch method takes one mandatory argument- the

URL of the resource that you want to get. It also accepts optional parameters like HTTP method, headers, body, and more.

Syntax:

```
fetch(url [, options])
.then(response => {    // Handle the response  })
.catch(error => {    // Handle any errors  });
```

Using Fetch API to Get Data

To Get data using the Fetch API in JavaScript, we use the **fetch() function** with the URL of the resource we want to fetch. By default, the fetch method makes the **Get request**.

Example: Get Data using Fetch API in JavaScript

The below examples show how to fetch data from a URL using JavaScript fetch API.

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => {
    console.log('Data received:', data);
```

```
  })
  .catch(error => {
    console.error('There was a problem with the fetch operation:', error);
  });

} catch (error) {
  console.error('Error:', error);
}
```

Explanation:

The code uses a try/catch block to handle errors during a POST request using the Fetch API. It sends data to ‘https://api.example.com/data’, with JSON formatting. If the response is not successful, it throws an error. The catch block logs any errors encountered during the process.

Making a Post request:

POST requests are used to send data to a server. This is commonly used when submitting forms or sending data to create a new resource. To use the JavaScript Fetch API to post data to a server, you can make a POST request with the desired data and handle the response.

Example: Post Data using Fetch API in JavaScript

Here, let’s assume we want to create a new user by

sending data to an API endpoint located at https://api.example.com/users. We specify this endpoint as the target URL for our POST request.

Within the fetch options, we set the **method property to 'POST'** to indicate that this is a POST request. Additionally, we include the **headers property** to specify that we are sending JSON data in the request body. The body of the request contains the user data, which is converted to JSON format using JSON.stringify().

```
// Data to be sent in the POST request (in JSON format)
const postData = {
  username: 'exampleUser',
  email: 'user@example.com'
};

// POST request options
const requestOptions = {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(postData)
};

// Make the POST request
fetch('https://api.example.com/users', requestOptions)
  .then(response => {
    // Check if the request was successful
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    // Parse the JSON response
    return response.json();
  })
  .then(data => {
```

```
  // Handle the data returned from the server
  console.log('Post request response:', data);
})
.catch(error => {
  // Handle any errors that occurred during the fetch
  console.error('There was a problem with the fetch operation:', error);
});
```

Explanation:

1. We define the data to be sent in the POST request, which is an object containing a username and email.
2. We specify the options for the POST request, including the HTTP method (POST), request headers (Content-Type: application/json), and request body (the postData object converted to JSON using JSON.stringify()).
3. We use the fetch() function to make a POST request to the specified URL (https://api.example.com/users) with the request options.
4. We handle the response using .then() and check if it's successful by accessing the response.ok property. If the response is okay, we parse the JSON response using response.json().
5. Once the JSON data is retrieved, we can then handle it accordingly.
6. Any errors that occur during the fetch operation are caught and handled using .catch().

Error Handling

When working with the Fetch API, a robust error-handling strategy is crucial, particularly for POST requests involving sensitive data. Utilize a try/catch block to encapsulate your code, with the catch() method

to manage errors

Example: Error handling in Fetch API

Below is an example how to handle errors while using fetch API.

```
try {
  const response = await fetch('https://api.example.com/data', {
    method: 'POST',
    body: JSON.stringify(data),
    headers: {
      'Content-Type': 'application/json'
    }
  });
  if (!response.ok) {
    throw new Error('Network response was not ok');
  }
  const result = await response.json();
  console.log('Response:', result);
} catch (error) {
  console.error('Error:', error);
}
```

Explanation:

The code uses a try/catch block to handle errors during a POST request using the Fetch API. It sends data to ‘https://api.example.com/data’, with JSON formatting. If the response is not successful, it throws an error. The catch block logs any errors encountered during the process.

Asynchronous JavaScript:

Asynchronous JavaScript is a programming approach that enables the non-blocking execution of tasks, allowing concurrent operations, improved responsiveness, and efficient handling of time-consuming operations in web applications, JavaScript is a single-threaded and synchronous language. The code is executed in order one at a time, But Javascript may appear to be asynchronous in some situations.

There are several methods that can be used to perform asynchronous javascript tasks, which are listed below:

Approach 1: Using callback

Callbacks are functions passed as arguments to be executed after an asynchronous operation completes. They are used in asynchronous JavaScript to handle responses and ensure non-blocking execution,

Syntax:

```
function myFunction(param1, param2, callback) {
    // Do some work...
    // Call the callback function
    callback(result);
}
```

Example: In this example, the myFunction simulates an async task with a 3s delay. It passes fetched data to the callback, which logs it. Output after 3s:

```
function myFunction(callback) {
    setTimeout(() => {
        const data = { name: "Aman", age: 21 };
        callback(data);
    }, 3000);
}

myFunction((data) => {
    console.log("Data:", data);
});
```

Output:

```
Data: { name: 'Aman', age: 21 }
```

Approach 2: Using Promises

Promises are objects representing the eventual completion (or failure) of an asynchronous operation, providing better handling of asynchronous code with .then() and .catch().

Syntax:

```
let promise = new Promise(function(resolve, reject){
     //do something
});
```

Example: In this example, The function mydata() returns a Promise that resolves with data after a delay. The data is logged, or an error is caught if rejected, after 2 seconds.

```
function mydata() {

    return new Promise((resolve, reject) => {
        setTimeout(() => {
            const data = { name: "Rohit", age: 23 };
            resolve(data);
        }, 2000);
    });
}

mydata()
    .then((data) => {
        console.log("Data:", data);
    })
    .catch((error) => {
        console.error("Error:", error);
    });
```

Output:

Data: { name: 'Rohit', age: 23 }

A callback is a function passed as an argument to another function

This technique allows a function to call another function

A callback function can run after another function has finished.

When to Use a Callback?

The examples above are not very exciting.

They are simplified to teach you the callback syntax.

Where callbacks really shine are in asynchronous functions, where one function has to wait for another function (like waiting for a file to load).

Functions running in **parallel** with other functions are called

asynchronous

A good example is JavaScript setTimeout().

Callback Alternatives:

With asynchronous programming, JavaScript programs can start long-running tasks, and continue running other tasks in paralell.

But, asynchronus programmes are difficult to write and difficult to debug.

Because of this, most modern asynchronous JavaScript methods don't use callbacks. Instead, in JavaScript, asynchronous programming is solved using **Promises** instead.

"Producing code" is code that can take some time

"Consuming code" is code that must wait for the result

A Promise is an Object that links Producing code and Consuming code.

```
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject();  // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

When the producing code obtains the result, it should call one of

the two callbacks:

When	Call
Success	myResolve(result value)
Error	myReject(error object)

async makes a function return a Promise

await makes a function wait for a Promise

Async Syntax:

The keyword async before a function makes the function return a promise:

Example:

```
async function myFunction() {
  return "Hello";
}
```

Is the same as:

```
function myFunction() {
  return Promise.resolve("Hello");
}
```

Await Syntax:

The await keyword can only be used inside an async function.

The await keyword makes the function pause the execution and wait for a resolved promise before it continues:

```
let value = await promise;
async function myDisplay() {
  let myPromise = new Promise(function(resolve, reject) {
```

```
    resolve("I love You !!");
  });
  document.getElementById("demo").innerHTML = await
myPromise;
}

myDisplay();
```