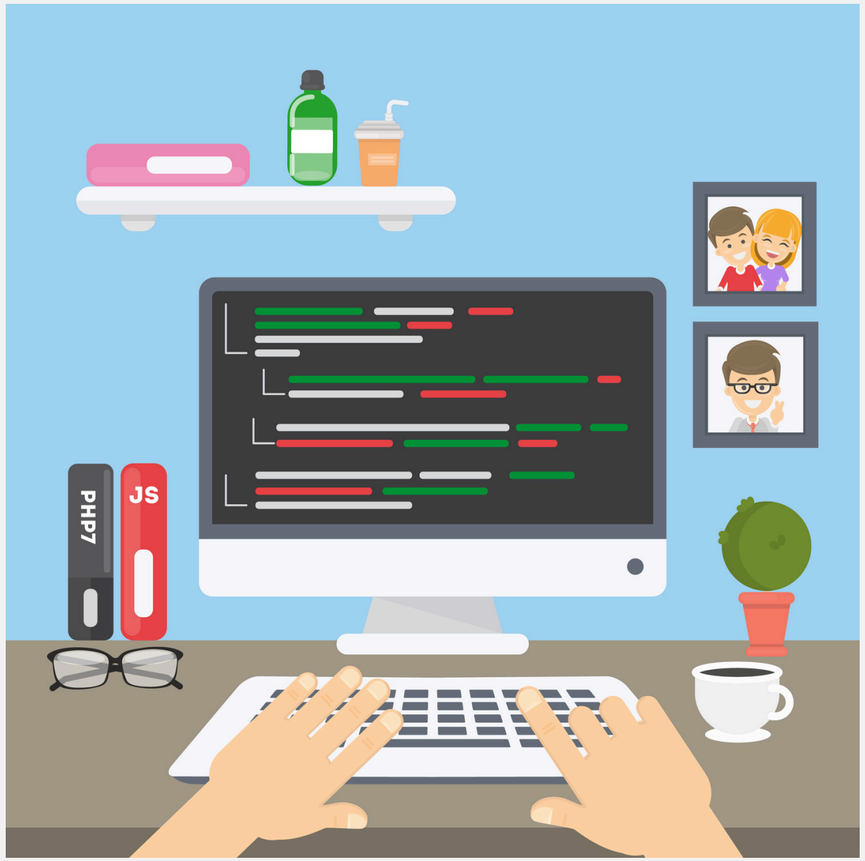
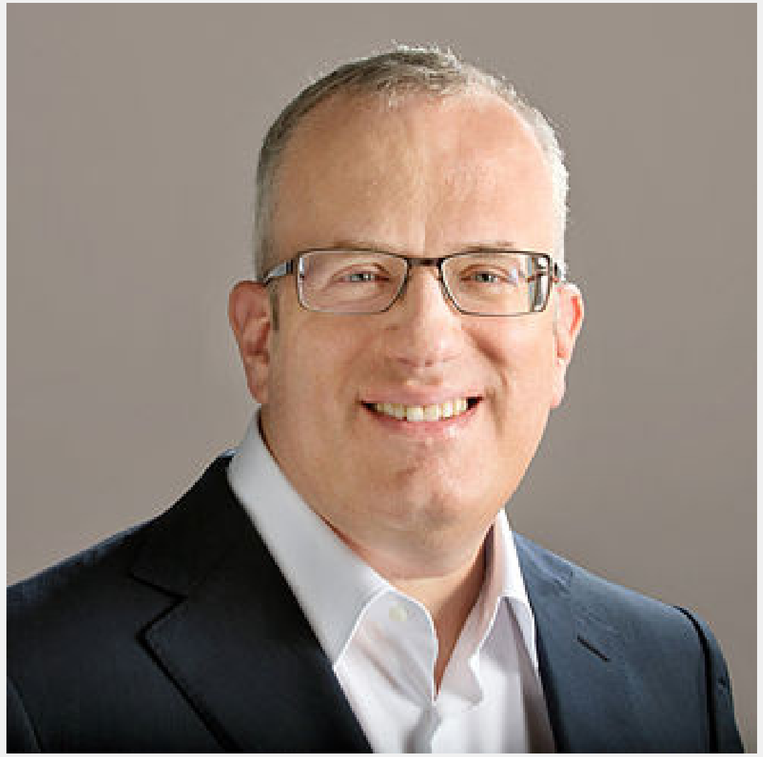


Introduction to JavaScript

"A Look into the World of Web Programming"

The Origins of JavaScript

- In 1995, all websites were static.
- Websites were made purely with HTML and CSS. Users could only view content without any interaction.
- **Brendan Eich** created JavaScript to make websites dynamic.



Fun Fact:

- JavaScript was created in just 10 days to bring interactivity to websites.
- Its name was inspired by the popular programming language Java, though the two are unrelated.

What is JavaScript?

Programming Language

- A programming language is a tool that helps us build logical abilities
- It allows us to perform tasks and solve problems.
- Example -> JavaScript, java, c++

OR

Scripting Language

- A scripting language automates manual tasks.
- It is used for the automation of repetitive tasks.
- Examples: JavaScript, Python, Ruby, PHP, bash.

Type of Programming Language

Programming languages are of two types

Interpreted Language

- Code is executed line by line by an interpreter. No need for a separate compilation step.
- **Example** -> JavaScript, python.

Compiled Language

- Code is first converted into machine code all at once by a compiler, and then the program runs.
- **Example**: C, C++, java.

How did the Browser work?

The browser works using two main engines

Layout Engine

- This engine is responsible for rendering or displaying the HTML and CSS of a webpage.

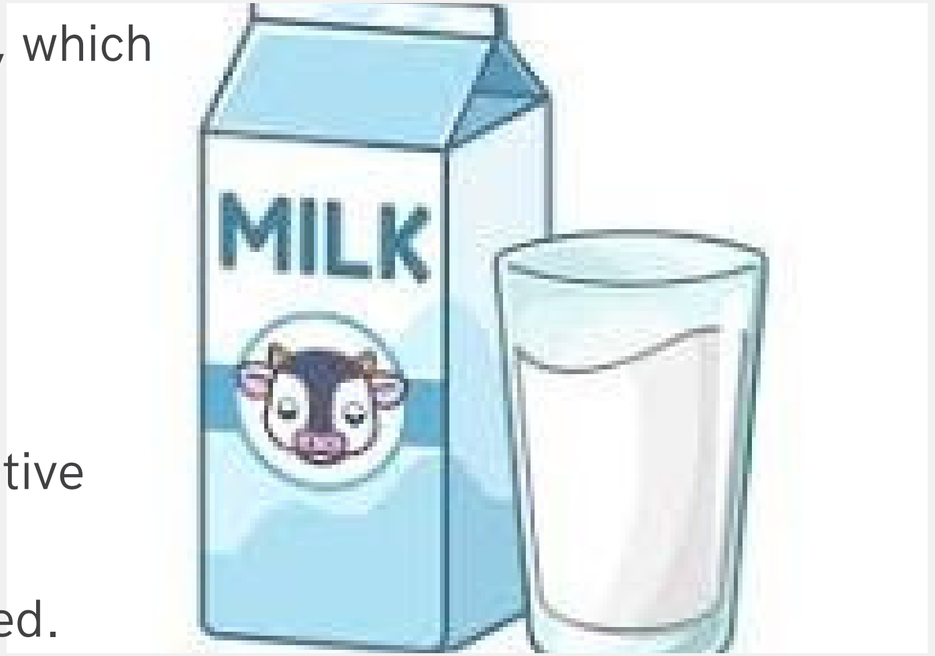
Js Engine

- This engine is used to run JavaScript code on the webpage.
- Popular JS engines include V8 (used by Chrome).

Data Types

Primitive Data Types:

Primitive types are the basic, pre-defined data types in JavaScript, which cannot be broken down into simpler types. Example: milk



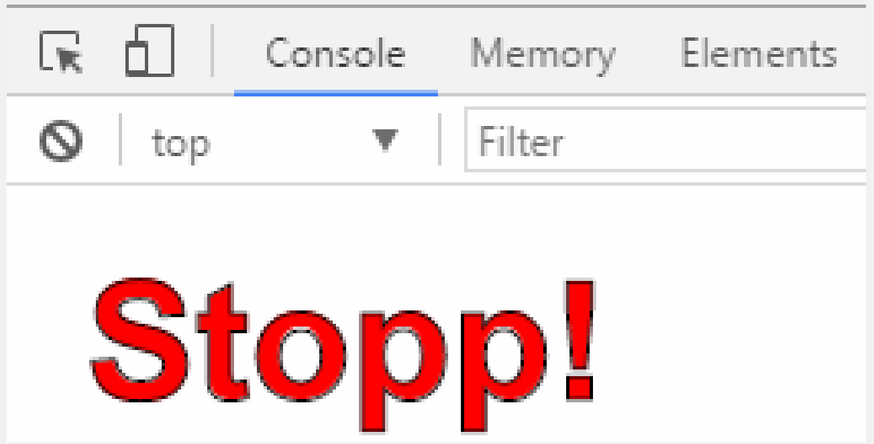
JavaScript has 7 primitive data types:

1. Boolean: Represents either true or false.
2. String: A sequence of characters, for example, "Hello".
3. Number: Includes integers, floating-point numbers, and negative numbers, like 5, 3.14, or -20.
4. Undefined: A variable that has been declared but not initialized.
5. Null: Represents an empty or non-existent value.
6. Symbol: A unique and immutable value used as an identifier.
7. BigInt: Allows for representing integers larger than the Number type can handle.

what is a console?

The console in JavaScript acts like a REPL (Read, Evaluate, Print, Loop). A REPL is an interactive environment where you can enter code, have it executed, and see the result immediately.

1. Read: The console reads the input (the code or expression you provide).
2. Evaluate: It then evaluates the input (executes the code or expression).
3. Print: After execution, the result of the evaluation is printed or displayed in the console.
4. Loop: After printing, it goes back to waiting for the next input, and the cycle continues.



What is a Variables?

- A variable is like a named storage location that uniquely identifies data in the program.
- Variables are references, or unique identifiers, that point to a specific location in memory.
- Variables act like buckets where you store data.
- JavaScript, the bucket doesn’t care about what type of data you put in it.
- Example -> water bottle.

How to Define a Variables?

We define variables using **declaratives:**

```
let myName = 'Rahul';
```

Yahan, myName ek variable hai, aur usme value 'Rahul' store hai.

String Data Type

- To show two strings we use (+) concatenate.
- to give space between two strings we give (" ").

```
// strings -write

// concatination -> writing multiple strings

let pehlaNaam = 'rahul';
let aakhariNaam = 'prajapati';

// let pooraNaam = pehlaNaam + aakhariNaam;

// let pooraNaam = pehlaNaam + " " + aakhariNaam;
```

- best way to add two strings and also give space between strings → backticks →
- jis chij ko evaluate karunga→ ${}

let pooraNaam = `my firstname is ${pehlaNaam} and lastname is ${aakhariNaam}`;
console.log(pooraNaam);

String Methods

- length→ find the length of the string
- toUpperCase()-> convert small letters into capital
- toLowerCase(). → convert capital letters into small
- trim(). → it removes extra spaces from a string

```
let naam2 = "      Prajapati      ";
 let ans = naam2.trim();
console.log(ans);
console.log(naam2.length);
```

Object

- It is an unordered data structure. → There is no order in the value given.
- A data structure is a way of organizing, storing, and managing data efficiently so that it can be accessed and modified easily.
- An object is a collection of properties.
- properties is a pair of key: value.

```
//syntax of object

let person = {
      //empty object
}
```

```
let person1 = {
      name : "rahul",
       age : 22,
       favColor : "white"
 }
console.log(person1);
```

Array

- It is an ordered Data structure. → jis order me aap value send karte ho usyi order me store bhi hota hai.
- It is not homogenous, which means it has the same data type.
- It is a heterogeneous data type (which having different types of primitive data types is possible)
- it is a zero-based indexing.

```
let arr = [1,2,3,4,5];
console.log(arr);
 console.log(arr[31]);
 let arr2 = [1,2,3,'sm' , 'Akshay' , 'neha' , 'sachin' , null , undefined , true];
console.log(arr2);
```

Array Methods

push()

- add at the end, array ke last me daal do.
- we can add multiple values at the same time.
- it returns something →its function will return the length of the array after adding the value at the end.

pop()

- remove an element from the last of the array, it will not take argument.
- we cannot remove multiple elements.
- its function will return the removed value

Array Methods

shift()

- it will remove an element from the start of the array.
- its function will return the value of removed from starting

unshift()

- It will add the value in the starting index of the array
- , and we can add multiple values.
- it returns the length of the array.

Loops

for-of

- it is used in an array. it is used for ordered data structure.
- item -> is a variable that is an iterator.

```
let arr = [10, 20, 30, 40,50]
for(let item of arr){
        console.log(item);
 }
```

for-in

- it works on unordered data structures.

```
let obj = {
first : 'rahul',
 last : 'prajapati',
age: 22,
 favColor : 'white',
 isMale : true
 }
```

Function

- it is a reusable code.
- what is the nature of a function → it always returns something.
  →v.v.v.v.imp
- Functions are the heart of Java script.
- it is a program, jo kuch kam krva raha hai , jab usye krne ko bole.

syntax of function

- reserved keyword that is a function keyword used to create a function
- calling of function is important.

```
function sum(){
        //working
 }
sum();
```

Parameterized functions:

- when the value of the argument passed during calling is catch as a parameter in the function.
- Argument → when we call a function and usme koyi value add krte hai.
- function ka aasli kam krna is returning something.

```
function sum2(){
    let num1 = 10;
    let num2 = 20;
    return (num1 + num2);
 }
// console.log(sum2());
 let ans = sum2();
console.log(ans);
```

first-class function?

- the tendency of a function to be able to point to the variable is known as a first-class function.
- jab kisi function ko kisi variable me se point karaya jata hai is called a first-class function.

```
let learn = function (){
console.log("hii i am learning")
 }
learn()
```

Advance object

- Method → A function that is defined under an object is called a method.

```
let anything= {
    a:10, b:20,
    c: true,
    fn: function sam(){
            return("Rahul object padha raha hai")
  }
 }
let ans = anything.fn();
console.log(ans);
```

Let, var, and const:

- There are three declarative used to declare variables.
- To declare means to formally define or introduce something, usually in programming
- Redeclare → declaring the same variable again.
- Reinitialize → reinitializing the same variable again with different data types.

Let:

- redeclare is not possible.
- reinitialize is possible.

Const:

- redeclare is not possible.
- reinitialize is not possible.

Var:

- both are possible.

How Js work:

1. whenever any JS code is run a **Global Execution Context is created**.
2. Inside that **GEC** we have two phases. **MCP AND CEP**.
3. MCP → The memory creation phase. and CEP → The code execution phase.
4. Every line of code runs inside the call stack.
5. MCP → is a phase that comes into the picture when no line of code is executed.
6. CEP → is a phrase that comes into the picture after the completion of MCP. In CEP code is executed line by line.
7. Whenever all the tasks of any execution context are done, the relevance of an execution context is negligible. Since all the tasks are done, this execution context will come out of the call stack.
8. the js code runs till the time your call stack is not empty.

Hoisting:

1. is a situation where you tried to excess your variables and your function before even declaring it.
2. hositing done in var, let, and const.
3. but there are two types of hoisting →
4. var → it will give **undefined**
5. let and const → give error →**dtz**

```
console.log(a)
fn()
var a = 100;
 function fn(){
    console.log("i am don jayant")
}
```

DTZ→ Dead Temporal Zone

1. is a phase between your MCP and CEP in this phase you are is aware of the variable's existence but since the zone is dead you are not allowed those variables.
2. it only occurs in let and const
3. dtz says that I know I have a variable but I cannot let you excess it before u initialize that.
4. ERROR → cannot access ‘b’ before initialization
5. let and const it give error → DTZ

```
console.log(a) fn();
 let a = 100;
const a = 100;
function fn(){
    console.log("i am don jayant");
 }
```

Scope:

we have 4 types of scope →

1. **global scope** → in the case of var, the variable is not excessing from function, the variable is stored in a global object.
2. **script scope** → when we create a variable with let and const which is not a part of block { }, the variable is assigned to script scope.
3. **functional scope / local scope→** is always stored in the global scope.
4. **block scope** → in case of let and const variable is present in block { } , the variable excess in block scope.

var:

for the var declarative we have two possible scopes.

- Global scoping: in this, the variable and function are declared in a global execution context.
- functional scoping / local scope→ if any variable is declared in functional scoping a new local scope is created in scope.

```
var a = 10;
function lol() {
    console.log(a);
}
console.log(a);
lol();
```

```
var x = 10;
function sam() {
    var x = 20;
    console.log(x);
}
sam();
console.log(x);
```

Let and Const::

we have other two scopes→ script and block.

1. if else, for loop → except function → block scope is created

- Script → when a variable a decelerated by let and const a script scope is created in GEC in which the variable is present.
- block → is nothing but a {curly brackets }
- block scope → in case of let and const variable is present in block { } , the variable excess in block scope.

```
let score = 100;
    if (score > 30) {
   let x = 20;
}
console.log(x);
```

```
let x = 10;
{
    let x = 20;
    console.log(x);
 }
console.log(x);
```

Higher Order Function

TWO SITUATIONS :

CASE 1:

- HOF are the functions that either accept the entire function in it as an argument/parameter.

CASE 2:

- HOF are the functions that return the entire function from it.

Callback Function

Callback functions are being sent to some other function (HOF) and are being executed as well.

- call karna zaruri hai, varna callback nhi hai.

```
function a(b){
    console.log("inside a") b();
 }
function b(){
     console.log("inside b") }
// a()
 a(b);
```

More Array Method

for-each()

- is a method used to iterate over the elements of an array.
- it does not return anything.
- it uses only two arguments.
- Item → iterate on the elements.
- index→ gives an index of the elements.
- it is a Higher-order-function and it accepts the callback function

```
let arr = [1,2, 3,4, 5]
arr.forEach(function(item , index){
    console.log(`${item} is at index:${index}`);
})
```

More Array Method

map()

- **map is a method and it also accepts a callback function**
- **It returns an entirely new array with the same number of elements as the original array.**

```
let marks = [10, 15, 18, 9, 28, 12, 5, 40];
  let newMarksArray = marks.map(function(item , index){
      return [ item*2 , index ];
})
console.log(marks);
 console.log(newMarksArray);
```

More Array Method

filter()

- The filter is a method and it also accepts a callback function
- it is used to filter out the true value (only true values)
- if values are not true then it gives an empty array
- it also returns the newArray but it might not equal the original array it can be a lesser number as well

```
let marks = [1,2,3,4,5,6,7,8,9,10];
 let filteredArray = marks.filter(function(item,index){
       if(item >= 5){
       console.log(index)
        return true;
 }
return false;
 })
 console.log(marks);
 console.log(filteredArray);
```

Constructor function

- when we call a function using a new keyword it returns the empty object and this is called a constructor function.
- it is used as blue print to generate objects with the same property.
- this keyword points to the newly created object. which is used to add properties an object

```
/function user(){
    this.username = 'rahul' ;
    this.email = 'rahul.india@gmail.com';
; }
 let user1 = new user();
 console.log(user1);
```

asynchronous vs synchronous

- js engine wait for none
- if we want to provide delay in code execution in js we can give by web Api called as setTimeOut() → it provide delay in code.
- It is a call back function and it takes function as argument and delay time in milisecond.
- It is not a part of js.
- It is part of Browser.
- these are called as web apis.

```
setTimeout() -> delay provide in our code
/it takes two arguments that is callBack function and delay time in millisec

console.log("first");

setTimeout( function(){ //cb func , delay in ms
    console.log("run after 4 seconds")
} , 4000 )

console.log("end");
```