History of JavaScript

JavaScript was created by Brendan Eich, a programmer at a company called Netscape Communications. Originally, JavaScript was named 'Mocha', but it quickly gained popularity as “LiveScript” and later evolved into 'JavaScript'.

Initially, JavaScript was designed solely to provide scripting capabilities aimed at enhancing web pages with dynamic content and interactivity. With the growth of HTML5 and CSS3, JavaScript's capabilities expanded further, enabling developers to construct sophisticated web applications and interactive user interfaces.

Many JavaScript frameworks and libraries, such as AngularJS, ReactJS, and Vue.js, have revolutionized frontend development. Additionally, Node.js, a powerful JavaScript runtime environment, has empowered developers to build server-side applications using JavaScript.

Introduction of JavaScript

JavaScript is a high-level, interpreted programming language widely used for web development. It enables interactive and dynamic content on websites, such as animations, form validations, and real-time updates. Following the ECMAScript standard, JavaScript supports multiple programming paradigms, including functional, event-driven, and object-oriented approaches. It runs not only in web browsers but also on servers using environments like Node.js. With built-in methods and APIs, JavaScript efficiently manipulates the Document Object Model (DOM) to update web pages dynamically. Modern features like async/await, promises, and modules make JavaScript more powerful and flexible. Additionally, it has a vast ecosystem, including popular libraries and frameworks like React, Angular, and Vue.js. JavaScript can also handle HTTP requests, interact with databases, and build full-stack applications. Due to its versatility and wide adoption, it remains one of the most important programming languages in the world.

Let , Var & Const

Variable declaration is one of the easiest tasks in every programming language. However, when it comes to JavaScript, it can become a bit tricky due to the special features of the language

Var

The var keyword was the traditional way of declaring variables in JavaScript. Variables declared with var are function-scoped or globally-scoped, but not block-scoped. This means that variables declared with var are hoisted to the top of their function or global scope. This hoisting behavior can sometimes lead to unexpected results and make debugging challenging.

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

example();   // 10
console.log(x);   // Throws ReferenceError: x is not defined
```

# Reassignment and Redefinition using var

In JavaScript, the var keyword allows variables to be both reassigned and redefined within the same scope, unlike let which only allows reassignment. This behavior of var can sometimes lead to unexpected results and bugs in code if not used carefully.
```

```
// Re assignment using var
var x = 10;
console.log(x);     // Output : 10

x = "Alaci";
console.log(x);     // Output : Alaci
```

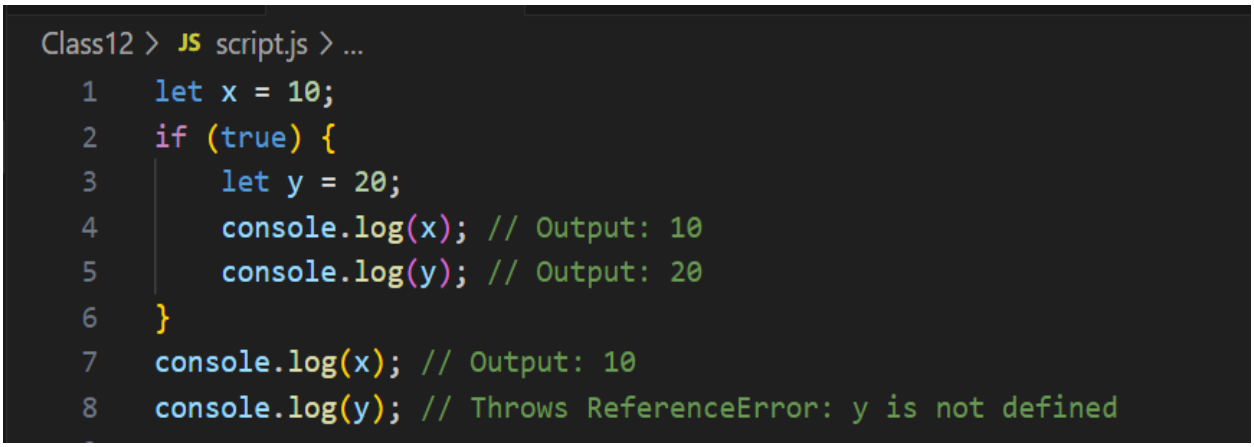
Class12 > JS script.js > ...

```
// Re definition using var
var x = 10;
console.log(x);     // Output : 10

var x = "Alaci";
console.log(x);     // Output : Alaci
```

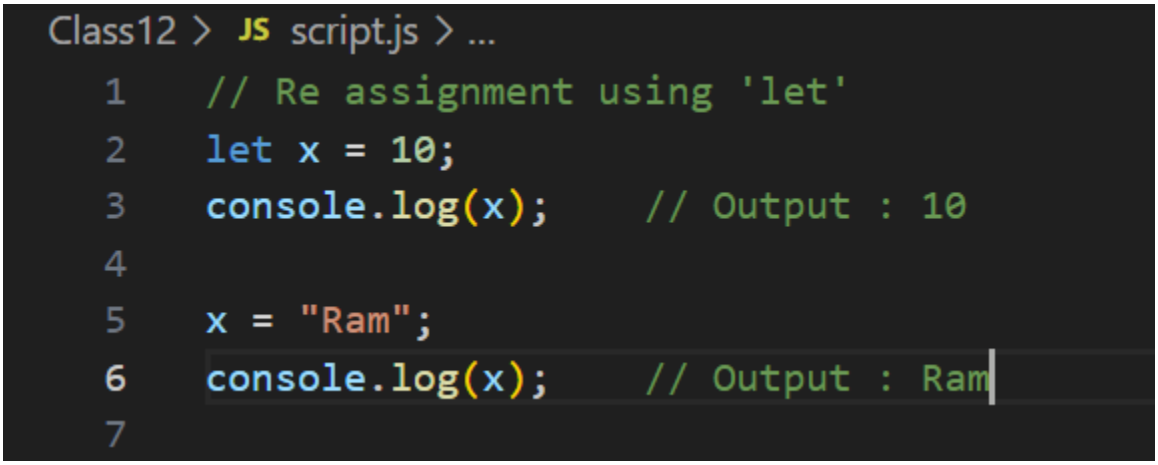
Let

In JavaScript, the let keyword is used to declare variables that are block-scoped, meaning they are only accessible within the block in which they are defined. Block scope refers to any code block delimited by curly braces {} such as loops, conditionals, or function bodies.



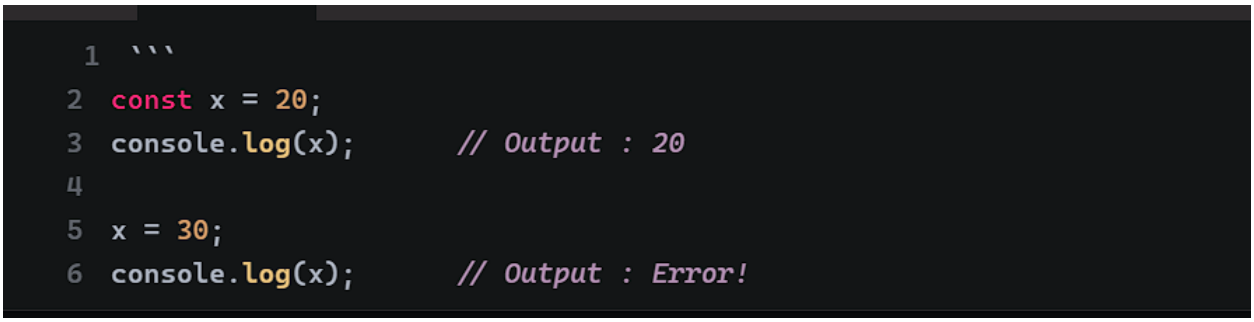
Reassignment and Redefinition using let

Variables declared with let can be reassigned, just like those declared with var, but they cannot be redeclared within the same block scope. This helps prevent accidental redeclaration of variables and can aid in writing more predictable and maintainable code.



Const

In JavaScript, the const keyword is used to declare constants. Constants are variables whose values cannot be reassigned once they are initialized. This means that once a value is assigned to a constant using const, it cannot be changed or reassigned throughout the execution of the script.



Constants declared with const must be initialized with a value. Unlike variables declared with var or let, which can be declared without immediately assigning a value, const requires initialization at the time of declaration.

Operators in JavaScript

JavaScript is a dynamic programming language, has various operators to perform operations on data and manipulate values.

Types of operators

1. Arithmetic Operators

- **Addition**
  The addition operator is denoted by '+'.
- **Subtraction**
  The subtraction operator is denoted by '-'.
- **Multiplication**
  The multiplication operator is denoted by '*'.
- **Exponentiation**
  he exponentiation operator is denoted by '**'.
- **Division**
  The division operator is denoted by '/'.
- **Modulus**
  The modulus operator is denoted by '%'.
- **Increment & decrement**
  Increment operator is denoted by '++', and the decrement operator is denoted by '--'.

2. Assignment Operators

The assignment operator assigns value to the variables. The most common assignment operator we use is '='.

For example:

```
let FirstName = "Maniraj";
let y = 4;

console.log(FirstName);     // Maniraj
 console.log(y);                  // 4
```

3. Logical Operators

There are typically three types of logical operators :

(a) AND (&&)

```
let alert1 = true;
let alert2 = false;
let alert3 = true;

console.log(alert1 && alert2);    // false
console.log(alert2 && alert3);    // false
console.log(alert1 && alert3);    // true
```

(b) OR (||)

```
let alert1 = false;
let alert2 = false;
let alert3 = true;

console.log(alert1 || alert2);    // false
console.log(alert2 || alert3);    // true
console.log(alert1 || alert3);    // true
```

(c) NOT (!)

```
let alert1 = false;
let isFirst = true;

console.log(!alert1);      // true
console.log(!isFirst);     // false
```

4. Comparison Operators

Comparison operators, such as greater then (>), less than (<), greater than or equal (>=), less than or equal (<=), equal to (==), not equal to (!=) are used to compare values and determine the relationship between them. These return a boolean value in the output.

If else statements

Conditional statements in JavaScript, such as 'if' and 'if-else', allow us to control the flow of our code based on certain conditions. With these statements, we can specify blocks of code to be executed only when certain conditions are met. For example, using an if statement, we can execute a piece of code if a particular condition evaluates to true. Alternatively, with an if-else statement, we can provide an alternative code block to be executed if the condition evaluates to false. These conditional statements provide powerful tools for building dynamic and responsive programs in JavaScript.

1. If Statement

In JavaScript, an 'if' statement is a fundamental conditional statement that allows us to execute a block of code only if the specified condition is true. If the condition is false, the code block associated with the 'if' statement is skipped.

Syntax:

```
if (condition){
    // Code block to be executed if the condition is true
}
```

2. If-else Statement

In JavaScript, an 'if-else' statement is a conditional statement that allows us to execute one block of code if a specified condition evaluates to true, and another block of code if the condition evaluates to false.

Syntax

```
if (condition) {
    // Code block to be executed if condition is true
} else {
    // Code block to be executed if condition is false
}
```

3. If-else-if Statement

An if-else-if statement is a conditional statement that allows us to test multiple conditions and execute different blocks of code based on the outcome of those conditions. It is an extension of the if-else statement and provides additional flexibility for handling multiple scenarios.

Syntax

```
if (condition1) {
    // Code block to be executed if condition1 is true
} else if (condition2) {
    // Code block to be executed if condition2 is true
} else {
    // Code block to be executed if all conditions are false
}
```

Loops in JavaScript

At the outset of learning to code in any specific language, grasping the concept of "loops" and their functionality is paramount. Loops are integral to maintaining the readability of our code and reducing pressure on the system by minimizing the number of lines needed to accomplish a given task. This

directly enhances the quality of the code, improves runtime efficiency, and saves considerable time that would otherwise be spent writing extensive code.

When discussing loops in computer programming, it's essential to note that there are primarily three types commonly used.

(a) For loop

Syntax

```
for ( initialization ; condition ; increment/decrement ) {
        // code block to be executed
}
```

(b) While loop

Syntax

```
while (condition) {
    // code to be executed
}
```

(c) Do-while loop

Syntax

```
do {
    // code to be executed
} while (condition);
```

JavaScript Math Object

The JavaScript Math object is a built-in object that provides mathematical constants and functions for performing mathematical operations in JavaScript. It provides a set of properties and methods for mathematical computations, making it easier to work with numbers in JavaScript.There are different types of function in Math Object.

1. Math.random()

For Example,

```
let value = Math.random();
console.log(value);
//Output : 0.048542301899791696

value = Math.random();
console.log(value);
//Output : 0.3692583661676796
```

2. Math.abs()

For Example,

```
let value = Math.abs(-2.7);
console.log(value);
//Output : 2.7
```

3. Math.sqrt()

For Example,

```
let value = Math.sqrt(15);
console.log(value);
//Output : 3.872983346207417

value = Math.sqrt(36);
console.log(value);
//Output : 6
```

4. Math.pow()

For Example,

```
let value = Math.pow(2,3);
console.log(value);
//Output : 8

value = Math.pow(4,2);
console.log(value);
//Output : 16
```

5. Math.min()

For Example,

```
let value = Math.min(2,3,70,9);
console.log(value);
//Output : 2

value = Math.min(-10,-3,-9);
console.log(value);
//Output : -10
```

6. Math.max()

For Example,

```
let value = Math.max(2,3,70,9);
console.log(value);
//Output : 70

value = Math.max(-10,-3,-9);
console.log(value);
//Output : -3

value = Math.max();
console.log(value);
//Output : -Infinity
```

7. Math.sin()

For Example,

```
let value = Math.sin(Math.PI/2);
console.log(value);
//Output : 1

value = Math.sin(Math.PI/6);
console.log(value);
//Output : 0.49999999999999994

value = Math.sin(0);
console.log(value);
//Output : 0
```

8. Math.cos()

For Example,

```
let value = Math.cos(Math.PI/3);
console.log(value);
//Output : 0.5000000000000001

value = Math.cos(Math.PI/6);
console.log(value);
//Output : 0.8660254037844387

value = Math.cos(0);
console.log(value);
//Output : 1
```

9. Math.tan()

For Example,

```
let value = Math.tan(Math.PI/3);
console.log(value);
//Output : 1.7320508075688767

value = Math.tan(Math.PI/6);
console.log(value);
//Output : 0.5773502691896257

value = Math.tan(0);
console.log(value);
//Output : 0
```

JavaScript Functions

A **function** in JavaScript is a reusable block of code that performs a specific task when called. It helps in organizing code, making it more readable and reusable. A function can take **parameters** (inputs) and return **a value** or execute a set of instructions.

Syntax

```
function  functionName(parameters) {
    // Code to execute
    return result; // (Optional)
}
```

Programs using Function in JavaScript

1. Multiplication Table

```
function multiplicationTable(num) {
    for (let i = 1; i <= 10 ; i++ ) {
        console.log(` ${num} x ${i} = ${num * i}` );
}
}
multiplicationTable(5);
```

2. Fibonacci Series

```
function fibonacciSeries(n) {
   let fib = [0, 1];
   for (let i = 2; i < n; i++) {
      fib[i] = fib[i - 1] + fib[i - 2];
   }
   console.log(fib.join(', '));
}
fibonacciSeries(10);
```

3. Check Prime Number

```
function isPrime(num) {
   if (num < 2) return false;
   for (let i = 2; i <= Math.sqrt(num); i++) {
      if (num % i === 0) return false;
   }
   return true;
}
console.log(isPrime(7)); // true
console.log(isPrime(10)); // false
```

4. Check Composite Number

```
function isComposite(num) {
   if (num < 2) return false;
   for (let i = 2; i < num; i++) {
      if (num % i === 0) return true;
   }
   return false;
}
console.log(isComposite(9)); // true
console.log(isComposite(7)); // false
```

5.Factorial of a Number

```
function factorial(n) {
   if (n === 0 || n === 1) return 1;
   return n * factorial(n - 1);
}
console.log(factorial(5)); // 120
```

6. Sum of Digits

```
function sumOfDigits(num) {
  let sum = 0;
  while (num > 0) {
    sum += num % 10;
    num = Math.floor(num / 10);
  }
  return sum;
}
console.log(sumOfDigits(1234)); // 10
```

7. Reverse a Number

```
function reverseNumber(num) {
  return parseInt(num.toString().split('').reverse().join(''));
}
console.log(reverseNumber(1234)); // 4321
```

8. Check Palindrome Number

```
function isPalindrome(num) {
  return num.toString() === num.toString().split('').reverse().join('');
}
console.log(isPalindrome(121)); // true
```

```
console.log(isPalindrome(123)); // false
```

9. Find GCD (Greatest Common Divisor)

```
function gcd(a, b) {
  return b === 0 ? a : gcd(b, a % b);
}
console.log(gcd(36, 60)); // 12
```

10. Check Armstrong Number

```
function isArmstrong(num) {
  let sum = 0;
  let temp = num;
  let digits = num.toString().length;

  while (temp > 0) {
    let digit = temp % 10;
    sum += Math.pow(digit, digits);
    temp = Math.floor(temp / 10);
  }
  return sum === num;
}
console.log(isArmstrong(153)); // true
console.log(isArmstrong(123)); // false
```

Event Listeners in JavaScript

Event listeners are a fundamental aspect of JavaScript programming, enabling developers to create dynamic and interactive web applications. They allow us to respond to user actions such as clicks, mouse movements, keyboard inputs, and more

What are 'Event Listeners' ?

In JavaScript, an event listener is a function that waits for a specific event to occur and then executes a callback function in response. Event listeners are commonly used to respond to user interactions such as clicks, mouse movements, keyboard inputs, or changes in the state of an HTML element.

How do Event Listeners Work?

Event listeners are attached to HTML elements, listening for specific events such as clicks, mouse movements, key presses, form submissions, etc. When the specified event occurs, the associated callback function is executed.

addEventListener()

In JavaScript, addEventListener() is a method used to attach an event handler to an element, enabling it to listen for specific events and execute code/callback function in response to those events.

Programs using Event Listeners in JavaScript

1. Button Click Event – Display Message

```
<!DOCTYPE html>
<html>
<head>
  <title>Button Click Event</title>
</head>
<body>
  <button id="btn">Click Me</button>
  <script>
    document.getElementById("btn").addEventListener("click", function() {
      document.write("Button Clicked!!! Welcome to JavaScript.");
    });
  </script>
</body>
</html>
```

2. Window Load Event – Show Welcome Message

```
<!DOCTYPE html> <html>
<head> <title>Window Load Event</title>  </head>
<body>
  <script>
    window.addEventListener("load", function() {
      document.write("Welcome! The page has fully loaded.");
    });
  </script>
```

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

3. Mouse Over Event – Change Text

```
<!DOCTYPE html>
<html>
<head>
  <title>Mouse Over Event</title>
</head>
<body>
  <p id="text">Hover over me!</p>
  <script>
    document.getElementById("text").addEventListener("mouseover", function() {
      document.write("You hovered over the text!");
    });
  </script>
</body>
</html>
```

4. Keypress Event – Display Key Pressed

```
<!DOCTYPE html>        <html>
<head>  <title>Keypress Event</title>  </head>
<body>
  <input type="text" id="inputBox" placeholder="Type something...">
  <script>
      document.getElementById("inputBox").addEventListener("keypress", function(event)
{
        document.write("You pressed: " + event.key);
```

```
    });
  </script>      </body>        </html>
```

4. Double Click Event – Change Message

```
<!DOCTYPE html>
<html>
<head>
  <title>Double Click Event</title>
</head>
<body>
  <button id="dblClickBtn">Double Click Me</button>
  <script>
    document.getElementById("dblClickBtn").addEventListener("dblclick", function() {
      document.write("You double-clicked the button!");
    });
  </script>
</body>
</html>
```

The End