JAVASCRIPT

INTRODUCTION

JavaScript is a high-level programming language used for web development. It runs on both the client-side and server-side and allows for interactive web pages by manipulating HTML and CSS. JavaScript provides programming capabilities, DOM manipulation, event handling, access to browser APIs, and has a vast ecosystem of libraries and frameworks. It is widely supported and enables the creation of dynamic and engaging web experiences.

HISTORY

JavaScript was created in the mid-1990s by Brendan Eich. It became popular for web scripting, underwent standardization, and introduced significant enhancements with ECMAScript versions. It expanded to server-side development with Node.js. JavaScript is now widely used for dynamic web experiences.

PURPOSE

JavaScript's purpose is to provide interactivity and dynamic functionality to web pages. It allows for real-time updates, interactive elements, and form validation. JavaScript can manipulate the DOM, access browser APIs, and is used for both client-side and server-side development. It enhances user experience and is essential for building web applications.

JavaScript Where to add

JavaScript code is typically embedded in HTML files or included as separate script files. It can be placed in the <script> tags within the HTML document, either in the <head> or <body> section.

The <script> Tag

In HTML, JavaScript code is inserted between <script> and <script> tags.

Example:

```
<script>
document.getElementById("demo").innerHTML = "First JS";
</script>
```

JavaScript in <head>

In this example, a JavaScript function is placed in the <head> section of an HTML page.
The function is invoked (called) when a button is clicked:

Example:

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
document.getElementById("tutorial").innerHTML = "nag palit ng paragraph.";
}
</script>
</head>
<body>
<h2>Tutorial JavaScript in head</h2>
<p id="tutorial">Paragraph</p>
<button type="button" onclick="myFunction()">click it</button>
</body>
</html>
```

JavaScript in <body>

In this example, a JavaScript function is placed in the <body> section of an HTML page.
The function is invoked (called) when a button is clicked:

Example:

```
<!DOCTYPE html>
<html>
<body>
<h2>Tutorial JavaScript in Body</h2>
<p id="tutorial">salita</p>
<button type="button" onclick="myFunction()">pindutin</button>
<script>
function myFunction() {
document.getElementById("tutorial").innerHTML = "salita changed.";
}
</script>
</body>
</html>
```

JavaScript Syntax

JavaScript follows a C-style syntax with statements and expressions.
It's case-sensitive and uses semicolons to separate statements.
Basic syntax includes variables, operators, functions, and control flow constructs like loops and conditionals.
JavaScript syntax is the set of rules, how JavaScript programs are constructed:

```
// How to create variables:
var x;
let y;

// How to use variables:
x = 5;
y = 6;
let z = x + y;
```

JavaScript Output

The console.log() function is commonly used for output in JavaScript.
It prints messages to the browser console, helping developers debug and understand the flow of their code.
JavaScript Display Possibilities
JavaScript can "display" data in different ways:
Writing into an HTML element, using innerHTML.
Writing into the HTML output using document.write().
Writing into an alert box, using window.alert().
Writing into the browser console, using console.log().

Using innerHTML
To access an HTML element, JavaScript can use the document.getElementById(id) method.
The id attribute defines the HTML element. The innerHTML property defines the HTML content:

Example:

```
<!DOCTYPE html>
<html>
<body>
<h1>Using innerHTML</h1>
<p>innerHTML</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
</body>
</html>
```

Using document.write()

For testing purposes, it is convenient to use document.write():

Example:

```
<!DOCTYPE html>
<html>
<body>
<h1>Using document.write()</h1>
<p>document.write()</p>
<script>
document.write(9 + 6);
</script>
</body>
</html>
```

Using window.alert()

Example:

```
<!DOCTYPE html>
<html>
<body>
<h1>Using window.alert()</h1>
<p>window.alert()</p>
<script>
window.alert(5 + 15);
</script>
</body>
</html>
```

Using console.log()

For debugging purposes, you can call the console.log() method in the browser to display data.

Example:

```
<!DOCTYPE html>
<html>
<body>
<script>
console.log(24 + 6);
</script>
</body>
</html>
```

JavaScript Variables

Variables are containers for storing data values. In JavaScript, you declare a variable using var, let, or const.

For example:

```
var x = 10;
let y = "Hello";
const z = x = y;
```

Example using var

```
var x = 5;
var y = 6;
var z = x + y;
```

Example using let

```
let x = 5;
let y = 6;
```

```
let z = x + y;
```

Example using const

```
const x = 5;
const y = 6;
const z = x + y;
```

Mixed Example

```
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
```

JavaScript Let

let is used to declare variables with block scope,
meaning they are limited to the block (pair of curly braces) in which they are defined.
It allows reassignment of values.

remember:

The let keyword was introduced in ES6 (2015)
Variables defined with let cannot be Redeclared
Variables defined with let must be Declared before use
Variables defined with let have Block Scope

Cannot be Redeclared
Variables defined with let can not be redeclared.
You can not accidentally redeclare a variable declared with let.

With let you can not do this:

```
let x = "John Doe";

let x = 0;
```

With var you can:

```
var x = "John Doe";
var x = 0;
```

Block Scope

Before ES6 (2015), JavaScript had Global Scope and Function Scope.
ES6 introduced two important new JavaScript keywords: let and const.
These two keywords provide Block Scope in JavaScript.
Variables declared inside a { } block cannot be accessed from outside the block:

Example:

```
{
let x = 2;
}
// x can NOT be used here
```

Variables declared with the var keyword can NOT have block scope.
Variables declared inside a { } block can be accessed from outside the block.

Example:

```
{
var x = 2;
}
// x CAN be used here
```

Redeclaring Variables

Redeclaring a variable using the var keyword can impose problems.
Redeclaring a variable inside a block will also redeclare the variable outside the block:

Example:

```
var x = 10;
```

```
// Here x is 10

var x = 2;

// Here x is 2
}
// Here x is 2
```

Redeclaring a variable using the let keyword can solve this problem.
Redeclaring a variable inside a block will not redeclare the variable outside the block:

Example:

```
let x = 10;
// Here x is 10
{
let x = 2;
// Here x is 2
}
// Here x is 10
```

JavaScript Const

const is used to declare variables with block scope,
like let, but its value cannot be reassigned once set.
It provides a way to create constants.

remember:

The const keyword was introduced in ES6 (2015)
Variables defined with const cannot be Redeclared
Variables defined with const cannot be Reassigned
Variables defined with const have Block Scope

Cannot be Reassigned
A const variable cannot be reassigned:

Example:
const PI = 3.141592653589793;
PI = 3.14; // This will give an error
PI = PI + 10; // This will also give an error

Must be Assigned
JavaScript const variables must be assigned a value when they are declared:
Correct
const PI = 3.14159265359;
Incorrect
const PI;
PI = 3.14159265359;

JavaScript Operators
JavaScript supports various operators for performing operations on variables and values.
Common ones include arithmetic operators (+, -, *, /),
comparison operators (==, ===, !=, !==), logical operators
(&&, ||, !), and assignment operators (=, +=, -=, *=, /=).

Arithmetic Operators

Operators Descriptios
+ Addition
- Subtraction
* Multiplication
** Exponentiation (ES2016)
/ Division
% Modulus (Division Remainder)

++ Increment
-- Decrement

Assignment Operators

Operators Example Same As
= x = y x = y
+= x += y x = x + y
-= x -= y x = x - y
*= x *= y x = x * y
/= x /= y x = x / y
%= x %= y x = x % y
**= x **= y x = x ** y

Comparison Operators

Operators Descriptios
== equal to
=== equal value and equal type
!= not equal
!== not equal value or not equal type
> greater than
< less than
>= greater than or equal to
<= less than or equal to
? ternary operator

Logical Operators

Operators Descriptios

&& logical and
|| logical or
! logical not

Type Operators

Operators Descriptios
typeof Returns the type of a variable
instanceof Return true if an objects is an instance of an object type

Bitwise Operators

Operators Descriptios Example Same as Result Decimal
& and 5 & 1 0101 & 0001 0001 1
| or 5 | 1 0101 | 0001 0101 5
~ not ~5 ~0101 1010 10
^ xor 5 ^ 1 0101 ^ 0001 0100 4
<< left shift 5 << 1 0101 << 1 1010 10
>> right shift 5 >> 1 0101 >> 1 0010 2
>>> unsigned right shift 5 >>> 1 0101 >>> 1 0010 2