JAVASCRIPT

JavaScript is the most powerful and versatile programming language used in the web.

It is a lightweight, cross-platform, single-threaded and interpreted (just in time complied) programming language.

It is a commonly used programming language to create dynamic and interactive elements in web applications.

It is easy to learn, compiled language.

Applications of JavaScript

Web Development: Adding interactivity and behavior to static sites JavaScript was invented to do this in 1995

Web Applications – like maps

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.

Eg.- node.js and bun.js

Games

Art - canvas

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

Mobile Applications:

Like – react native is capable of building cross platform mobile apps

Linking java script in html

Internal JS:

We can add JavaScript directly to our HTML file by writing the code inside the <script> tag. The <script> tag can either be placed inside the <head> or the <body> tag according to the requirement.

<script>  
 // JavaScript Code  
</script>

External JS:

We can write JavaScript code in another files having an extension.js and then link this file inside the <head> tag of the HTML file in which we want to add this code.

<!DOCTYPE html>

<html lang="en">

<head>

<title>

Basic Example to Describe JavaScript

</title>

</head>

<body>

<!-- JavaScript code can be embedded inside

head section or body section -->

<script>

console.log("Welcome to GeeksforGeeks");

</script>

</body>

</html>

Variables =>

a variable is a named container for storing data values.

Variables are used to hold various types of data, such as numbers, strings, objects, or functions.

Variable can be declare with one of following keywords :-

1. Var
2. Let
3. Const

Ex.- var a;

Note - semicolons are optional in js

Variable naming rules –

In JavaScript, variable names, also known as identifiers, must follow certain rules:

1. **Start with a letter, underscore (\_), or dollar sign ($)**: Variable names must begin with a letter, an underscore, or a dollar sign. They cannot start with a number.
2. **Subsequent characters can include letters, numbers, underscores, or dollar signs**: After the first character, variable names can contain letters (uppercase or lowercase), numbers, underscores (\_), or dollar signs ($). Spaces and special characters (except underscore and dollar sign) are not allowed.
3. **JavaScript variable names are case-sensitive**: This means that **myVariable**, **MyVariable**, and **myvariable** are all considered different variables.
4. **Avoid using reserved words**: JavaScript has reserved words that have special meaning in the language, such as **var**, **let**, **const**, **function**, **if**, **else**, etc. These words cannot be used as variable names.
5. **Use meaningful names**: Choose variable names that accurately describe the data they hold or the purpose they serve. This helps make your code more readable and maintainable.

Here are some examples of valid variable names in JavaScript:

var myVariable;

var \_myVariable;

var $myVariable;

var myVariable2;

var firstName;

var lastName;

var camelCaseVariable;

And here are some examples of invalid variable names:

var 123variable; // Cannot start with a number

var my variable; // Cannot contain space

var my-variable; // Cannot contain hyphen

var let; // Reserved word

var if; // Reserved word

The main differences between **var**, **let**, and **const** in JavaScript lie in their scope, hoisting behavior, and mutability. Here's a breakdown with examples:

1. **var**:
   * **var** declarations are function-scoped or globally scoped, but not block-scoped.
   * They are hoisted to the top of their function or global scope, which means they can be used before they are declared.
   * Variables declared with **var** can be re-declared and reassigned.

Example:

function example() {

console.log(x); // undefined

var x = 10;

console.log(x); // 10

}

1. **let**:

* **let** declarations are block-scoped, which means they are only accessible within the block in which they are defined (typically denoted by **{}**).
* They are not hoisted to the top of their block.
* Variables declared with **let** can be updated, but not re-declared within the same scope.

Example:

function example() {

console.log(x); // ReferenceError: Cannot access 'x' before initialization

let x = 10;

console.log(x); // 10

}

1. **const**:

* **const** declarations create constants, which are block-scoped just like variables declared with **let**.
* They are not hoisted to the top of their block.
* Constants must be initialized during declaration and cannot be reassigned a new value. However, if the value is an object or array, its properties or elements can be modified.

Example:

function example() {

const x = 10;

// x = 20; // TypeError: Assignment to constant variable

console.log(x); // 10

}

**JavaScript Data Types**

JavaScript is a **dynamically typed** (also called loosely typed) scripting language. In JavaScript, variables can receive different data types over time.

**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.

1. Number
2. String
3. Boolean
4. Null
5. Undefined
6. Symbol
7. BigInt

**Non-Primitive Data Types:**

1. Object

2. Array ( it’s also an object)

**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';

**Boolean:**

The boolean type has only two values: true and false. This data type is used to store yes/no values: true means “yes, correct”, and false means “no, incorrect”.

let isCoding = true; // yes  
 let isOld = false; // no

**NULL AND UNDEFINED :**

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("XYZ")  
let symbol2 = Symbol("XYZ")  
   
// Each time Symbol() method   
// is used to create new global Symbol  
console.log(symbol1 == symbol2); // False

**BigInt:**

Represents integers with arbitrary precision. It allows you to work with numbers beyond the safe integer limit for regular numbers.

Example:

let bigBin = BigInt("0b1010101001010101001111111111111111");  
// 11430854655n  
console.log(bigBin);

Object:

Represents a collection of key-value pairs (properties and methods).Example:

javascript

const person = {

firstName: 'John',

lastName: 'Doe',

age: 30,

sayHello: function() {

console.log('Hello, my name is ' + this.firstName);

}

};

**Array :-**

In JavaScript, an array is a special type of object used to store multiple values in a single variable.

Arrays can hold values of any data type, including other arrays, objects, and functions. They are particularly useful for storing lists of items or collections of related data.

Here's how you can create and access arrays in JavaScript:

Creating Arrays:

You can create an array using array literal syntax, which consists of square brackets [], and you can optionally initialize it with values:

let fruits = ['Apple', 'Banana', 'Orange'];

let numbers = [1, 2, 3, 4, 5];

let mixedArray = [1, 'two', true, { key: 'value' }];

Accessing Elements:

You can access individual elements of an array using square bracket notation, with the index starting at 0:

console.log(fruits[0]); // 'Apple' console.log(numbers[2]); // 3

**Operators : -**

JavaScript includes a variety of operators that you can use to perform different kinds of operations on operands.

**1. Arithmetic Operators**

Arithmetic operators are used to perform mathematical calculations.

let x = 10, y = 3;

console.log(x + y); // Addition: 13

console.log(x - y); // Subtraction: 7

console.log(x \* y); // Multiplication: 30

console.log(x / y); // Division: 3.3333...

console.log(x % y); // Modulus (remainder): 1

console.log(x \*\* y); // Exponentiation (x to the power of y): 1000

**2. Assignment Operators**

Assignment operators assign values to JavaScript variables.

let a = 10;

a += 5; // a = a + 5: 15

a -= 2; // a = a - 2: 13

a \*= 2; // a = a \* 2: 26

a /= 4; // a = a / 4: 6.5

a %= 3; // a = a % 3: 0.5

a \*\*= 2; // a = a \*\* 2: 0.25

**3. Comparison Operators**

Comparison operators are used to compare two values.

let a = 5, b = 10;

console.log(a == b); // Equal: false

console.log(a === b); // Strict equal: false

console.log(a != b); // Not equal: true

console.log(a !== b); // Strict not equal: true

console.log(a < b); // Less than: true

console.log(a > b); // Greater than: false

console.log(a <= b); // Less than or equal to: true

console.log(a >= b); // Greater than or equal to: false

**4. Logical Operators**

Logical operators are typically used with Boolean (logical) values.

[JavaScript Logical Operators](https://www.geeksforgeeks.org/javascript-logical-operators/) perform logical operations: AND (&&), OR (||), and NOT (!), evaluating expressions and returning boolean values.

| **Name** | **Description** | **Syntax** |
| --- | --- | --- |
| [**Logical AND (&&)**](https://www.geeksforgeeks.org/and-logical-operator-in-javascript/). | It checks whether two operands are non-zero (0, false, undefined, null, or “” are considered as zero), if yes then return the last operand when evaluating from left to right | Y = 5 and X = 6 Y && X is 6. |
| [**Logical OR (||)**](https://www.geeksforgeeks.org/or-logical-operator-in-javascript/) | It checks whether two operands are non-zero (0, false, undefined, null, or “” is considered as zero), if yes then return the first operand when evaluating from left to right. | Y = 5 and X = 0 Y || X is 5. |
| [**Logical NOT (!)**](https://www.geeksforgeeks.org/not-logical-operator-injavascript/) | It reverses the boolean result of the operand (or condition). | Y = 5 and X = 0 !(Y || X) is false. |

console.log(true && false); // Logical AND: false

console.log(true || false); // Logical OR: true

console.log(!true); // Logical NOT: false

**5. String Operators**

In JavaScript, the + operator can also concatenate strings.

let firstName = "John";

let lastName = "Doe";

console.log(firstName + " " + lastName); // Output: "John Doe"

**6. Conditional (Ternary) Operator**

The conditional operator assigns a value to a variable based on a condition.

let age = 18;

let eligibility = (age >= 18) ? "Can vote" : "Cannot vote";

console.log(eligibility); // Output: "Can vote"

**7. typeof Operator**

The typeof operator is used to determine the type of a variable or an expression.

console.log(typeof 42); // "number"

console.log(typeof 'Hello'); // "string"

console.log(typeof true); // "boolean"

console.log(typeof undefined); // "undefined"

console.log(typeof null); // "object" (this is considered a bug in JavaScript)

console.log(typeof {}); // "object"

console.log(typeof []); // "object" (array is a type of object)

console.log(typeof function() {}); // "function"

**Unary Operators**

**Increment (++)**

Adds one to its operand. The increment operator can be pre or post, which affects the value returned.

let a = 5;

console.log(a++); // Output: 5 (post-increment: returns the original value before incrementing)

console.log(++a); // Output: 7 (pre-increment: increments first, then returns the incremented value)

**Decrement (--)**

Subtracts one from its operand. Like the increment operator, the decrement operator can be pre or post.

let b = 5;

console.log(b--); // Output: 5 (post-decrement: returns the original value before decrementing)

console.log(--b); // Output: 3 (pre-decrement: decrements first, then returns the

**coercion in js (automatic type conversion)**

In the context of programming, specifically in JavaScript, "coercion" refers to the automatic or implicit conversion of values from one data type to another.

This happens because JavaScript is a loosely typed language, meaning variables don't have types; rather, the values have types.

**1. String and Number Coercion**

When you try to use the + operator with a string and a number, JavaScript will convert the number to a string and then concatenate them.

let number = 5;

let text = "The number is ";

let result = text + number; // The number is converted to a string

console.log(result); // Output: "The number is 5"

**2. Equality Coercion with ==**

Using == (double equals) allows coercion, while === (triple equals) does not, enforcing a check on both value and type.

let num = 0;

let str = "0";

console.log(num == str); // Output: true (number 0 is coerced to string "0")

console.log(num === str); // Output: false (type and value are not the same)

**3. Numeric String to Number Coercion**

When using numeric operators (except +, which concatenates if one operand is a string), JavaScript coerces string representations of numbers into numbers.

javascript

Copy code

let a = "5";

let b = "2";

console.log(a \* b); // Output: 10 (both strings are coerced to numbers)

console.log(a - b); // Output: 3

console.log(a / b); // Output: 2.5

**4. Coercion in Array Context**

When an array is involved in a coercion with a string, the array is converted to a string representation, which is essentially the elements of the array joined by commas.

let array = [1, 2, 3];

let string = " is an array";

console.log(array + string); // Output: "1,2,3 is an array"

## **5.Boolean to Number**

When a Boolean is added to a Number, the Boolean value is converted to a number

// The Boolean value true is

// converted to number 1 and

// then operation is performed

let x = true + 2;

// The Boolean value false is

// converted to number 0 and

// then operation is performed

let y = false + 2;

console.log(x);//3

console.log(y);//2