What is JavaScript

JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document.

JavaScript is an object-based scripting language which is lightweight and cross-platform.

**Difference between client-side scripting and server-side scripting :**

## Key Differences Between Server-side Scripting and Client-side Scripting

1. Server-side scripting is used at the backend, where the source code is not viewable or hidden at the client side (browser). On the other hand, client-side scripting is used at the front end which users can see from the browser.
2. When a server-side script is processed it communicates to the server. As against, client-side scripting does not need any server interaction.
3. The client-side scripting language involves languages such as HTML, CSS and JavaScript. In contrast, programming languages such as PHP, ASP.net, Ruby, ColdFusion, Python, C#, Java, C++, etc.
4. Server-side scripting is useful in customizing the web pages and implement the dynamic changes in the websites. Conversely, the client-side script can effectively minimize the load to the server.
5. Server-side scripting is more secure than client-side scripting as the server side scripts are usually hidden from the client end, while a client-side script is visible to the users.

| **Client-side scripting** | **Server-side scripting** |
| --- | --- |
| Source code is visible to the user. | Source code is not visible to the user because its output  of server-sideside is an HTML page. |
| Its main function is to provide the requested output to the end user. | Its primary function is to manipulate and provide access to the respective database as per the request. |
| It usually depends on the browser and its version. | In this any server-side technology can be used and it does not  depend on the client. |
| It runs on the user’s computer. | It runs on the webserver. |
| There are many advantages linked with this like faster.  response times, a more interactive application. | The primary advantage is its ability to highly customize, response  requirements, access rights based on user. |
| It does not provide security for data. | It provides more security for data. |
| It is a technique used in web development in which scripts run on the client’s browser. | It is a technique that uses scripts on the webserver to produce a response that is customized for each client’s request. |
| HTML, CSS, and javascript are used. | PHP, Python, Java, Ruby are used. |
| No need of interaction with the server. | It is all about interacting with the servers. |

# JavaScript Variable

1. [JavaScript variable](https://www.javatpoint.com/javascript-variable)
2. [JavaScript Local variable](https://www.javatpoint.com/javascript-variable#local)
3. [JavaScript Global variable](https://www.javatpoint.com/javascript-variable#gloabl)

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable

JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

**<script>**

function abc(){

var x=10;//local variable

}

**</script>**

## JavaScript global variable

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

<html>

<body>

<script>

var data=200;//gloabal variable

function a(){

document.writeln(data);

}

function b(){

document.writeln(data);

}

a();//calling JavaScript function

b();

</script>

</body>

</html>

<html>

<body>

<script>

function m(){

window.value=100;//declaring global variable by window object

}

function n(){

alert(window.value);//accessing global variable from other function

}

m();

n();

</script>

</body>

* So, in javascript the whole document is **global scope** and all the other functions and variables are contained in this global scope.
* Another is the **Local Scope**, variables declared inside the functions are considered to be of the local scope and it is futher divided into function scoped and block scoped.
* **Function Scope**: When a variable is declared inside a function, it is only accessible within that function and cannot be used outside that function.
* **Block Scope**: A variable when declared inside the if or switch conditions or inside for or while loops, are accessible within that particular condition or loop. To be consise the variables declared inside the curly braces are called as within block scope.
* **var** is called as function scope that is if a variable is declared using var keyword it will be accessible throughout the function.
* **let & const** are also called as block scope that is they are accessible within that particular block. Here let is used for variable which can be changed as we proceed through the program whereas const is used for a variable which doesn’t change till the program ends, that is it remains constant throughout the program

## Function Scope

The function scope is the accessibility of the variables defined inside a function, these variables cannot be accessed from any other function or even outside the function in the main file.

**For example**  
First, we created a variable inside a function and accessed it inside the function:

function abc() {  
  
year = 2021;  
  
// the "year" variable can be accessed inside this function  
  
console.log("The year is "+ Year);  
}  
  
// the "year" variable cannot be accessed outside here  
  
abc();

[**let**](https://www.geeksforgeeks.org/javascript-let/) **keyword in JavaScript:** The *let* keyword is an improved version of the *var* keyword.

**Scope:** [**block scoped:**](https://www.geeksforgeeks.org/javascript-es2015-block-scoping/) The scope of a *let* variable is only block scoped. It can’t be accessible outside the particular block ({block}). Let’s see the below example.

**Example 1:** The output is shown in the console.

|  |  |
| --- | --- |
| <script>      let a = 10;      function f() {          let b = 9          console.log(b);          console.log(a);      }      f();  </script>  **Example 2:** The code returns an error because we are accessing the *let* variable outside the function block. The output is shown in the console.   |  | | --- | | <script>      let a = 10;      function f() {          if (true) {              let b = 9                // It prints 9              console.log(b);          }            // It gives error as it          // defined in if block          console.log(b);      }      f()        // It prints 10      console.log(a)  </script> |   **Output:**  9 |

**Output: Example 4:** Users can declare the variable with the same name in different blocks using the *let* keyword.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| <script>    let a = 10    if (true) {      let a=9      console.log(a) // It prints 9    }    console.log(a) // It prints 10  [**const**](https://www.geeksforgeeks.org/javascript-const/) **keyword in JavaScript:** The *const* keyword has all the properties that are the same as the *let* keyword, except the user cannot update it.  **Scope:** [block scoped:](https://www.geeksforgeeks.org/javascript-es2015-block-scoping/) When users declare a *const* variable, they need to initialize it, otherwise, it returns an error. The user cannot update the *const* variable once it is declared.  **Example 1:** We are changing the value of the const variable so that it returns an error. The output is shown in the console.   |  | | --- | | <script>      const a = 10;      function f() {          a = 9          console.log(a)      }      f();  </script> |   **Output:**  a=9  TypeError:Assignment to constant variable.  **Example 2:** Users cannot change the properties of the *const* object, but they can change the value of properties of the *const* object.   |  | | --- | | <script>      const a = {          prop1: 10,          prop2: 9      }        // It is allowed      a.prop1 = 3        // It is not allowed      a = {          b: 10,          prop2: 9      }  </script> |   **Output:**  Uncaught SyntaxError:Unexpected identifier  </script> **Differences between var, let, and const**   |  |  |  | | --- | --- | --- | | **var** | **let** | **const** | | The scope of a *var* variable is functional scope. | The scope of a *let* variable is block scope. | The scope of a *const* variable is block scope. | | It can be updated and re-declared into the scope. | It can be updated but cannot be re-declared into the scope. | It cannot be updated or re-declared into the scope. | | It can be declared without initialization. | It can be declared without initialization. | It cannot be declared without initialization. | | It can be accessed without initialization as its default value is “undefined”. | It cannot be accessed without initialization otherwise it will give ‘referenceError’. | It cannot be accessed without initialization, as it cannot be declared without initialization. | | hoisting done , with initializing as ‘default’ value | Hoisting is done , but not initialized (this is the reason for error when we access the let variable before declaration/initialization | Hoisting is done, but not initialized (this is the reason for error when we access the const variable before declaration/initialization | |

**Output:**

9

10

9

10

**Example 1:** Variable ‘a’ is declared globally. So, the scope of the variable ‘a’ is global, and it can be accessible everywhere in the program. The output shown is in the console.

|  |
| --- |
| <script>      var a = 10          function f(){              console.log(a)          }      f();      console.log(a);  </script> |

**Output:**

10

10

</html>

# Javascript Data Types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

1. var a=40;//holding number
2. var b="Rahul";//holding string
3. JavaScript primitive data types
4. There are five types of primitive data types in JavaScript. They are as follows:

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Description** | |
| String | represents sequence of characters e.g. "hello" | |
| Number | represents numeric values e.g. 100 | |
| Boolean | represents boolean value either false or true | |
| Undefined | represents undefined value | |
| Null | represents null i.e. no value at all | |
| Data Type | Description | |
| Object | represents instance through which we can access members | |
| Array | represents group of similar values | |
| RegExp | represents regular expression | |
| Null | represents null i.e. no value at all |
| Data Type | Description |
| Object | represents instance through which we can access members |
| Array | represents group of similar values |
| RegExp | represents regular expression |

### 1. What are the different data types present in javascript?

To know the type of a JavaScript variable, we can use the **typeof** operator.

**1. Primitive types**

**String** - It represents a series of characters and is written with quotes. A string can be represented using a single or a double quote.

Example :

var str = "Vivek Singh Bisht"; //using double quotes

var str2 = 'John Doe'; //using single quotes

* **Number** - It represents a number and can be written with or without decimals.

Example :

var x = 3; //without decimal

var y = 3.6; //with decimal

* **BigInt** - This data type is used to store numbers which are above the limitation of the Number data type. It can store large integers and is represented by adding “n” to an integer literal.

Example :

var bigInteger = 234567890123456789012345678901234567890;

* **Boolean** - It represents a logical entity and can have only two values : true or false. Booleans are generally used for conditional testing.

Example :

var a = 2;

var b = 3;

var c = 2;

(a == b) // returns false

(a == c) //returns true

* **Undefined** - When a variable is declared but not assigned, it has the value of undefined and it’s type is also undefined.

Example :

var x; // value of x is undefined

var y = undefined; // we can also set the value of a variable as undefined

* **Null** - It represents a non-existent or a invalid value.

Example :

var z = null;

* **Symbol** - It is a new data type introduced in the ES6 version of javascript. It is used to store an anonymous and unique value.

Example :

var symbol1 = Symbol('symbol');

* typeof **of primitive types** :

typeof "John Doe" // Returns "string"

typeof 3.14 // Returns "number"

typeof true // Returns "boolean"

typeof 234567890123456789012345678901234567890n // Returns bigint

typeof undefined // Returns "undefined"

typeof null // Returns "object" (kind of a bug in JavaScript)

typeof Symbol('symbol') // Returns Symbol

**2. Non-primitive types**

* Primitive data types can store only a single value. To store multiple and complex values, non-primitive data types are used.
* Object - Used to store collection of data.
* Example:

// Collection of data in key-value pairs

var obj1 = {

x: 43,

y: "Hello world!",

z: function(){

return this.x;

}

}

// Collection of data as an ordered list

var array1 = [5, "Hello", true, 4.1];

There are following types of operators in JavaScript.

|  |  |
| --- | --- |
| Operator | Description |
| (?:) | Conditional Operator returns value based on the condition. It is like if-else. |
| typeof | checks the type of object. |

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

# JavaScript If-else

The **JavaScript if-else statement** is used to execute the code whether condition is true or false. There are three forms of if statement in JavaScript.

1. If Statement
2. If else statement
3. if else if statement

### JavaScript If statement

It evaluates the content only if expression is true. The signature of JavaScript if statement is given below

<html>

<body>

<script>

var a=20;

if(a>10){

document.write("value of a is greater than 10");

}

</script>

</body>

</html>

<script>

var a=20;

if(a%2==0){

document.write("a is even number");

}

else{

document.write("a is odd number");

}

</script>

# JavaScript Switch

The **JavaScript switch statement** is used to execute one code from multiple expressions. It is just like else if statement that we have learned in previous page. But it is convenient than if..else..if because it can be used with numbers, characters etc.

The switch statement is used to perform different actions based on different conditions.

The JavaScript Switch Statement

Use the switch statement to select one of many code blocks to be executed.

This is how it works:

* The switch expression is evaluated once.
* The value of the expression is compared with the values of each case.
* If there is a match, the associated block of code is executed.
* If there is no match, the default code block is executed.

## The break Keyword

When JavaScript reaches a break keyword, it breaks out of the switch block.

This will stop the execution inside the switch block.

It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

**Note:**If you omit the break statement, the next case will be executed even if the evaluation does not match the case.

## The default Keyword

The default keyword specifies the code to run if there is no case match:

// multiple case switch program

let fruit = 'apple';

switch(fruit) {

case 'apple':

case 'mango':

case 'pineapple':

console.log(`${fruit} is a fruit.`);

break;

default:

console.log(`${fruit} is not a fruit.`);

break;

}

<script type = "text/javascript">

// JavaScript program to illustrate switch-case

let i = 9;

switch (i)

{

case 0:

console.log("i is zero.");

break;

case 1:

console.log("i is one.");

break;

case 2:

console.log("i is two.");

break;

default:

console.log("i is greater than 2.");

}

</script>

# JavaScript Loops

The **JavaScript loops** are used to iterate the piece of code using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop
4. for-in loop