JAVASCRIPT

Why Study JavaScript?

JavaScript is one of the **3 languages** all web developers **must** learn:

1. **HTML** to define the content of web pages
2. **CSS** to specify the layout of web pages
3. **JavaScript** to program the behavior of web pages

There are many useful **Javascript frameworks** and libraries available:

- Angular
- React
- jQuery
- Vue.js
- Ext.js
- Ember.js
- Meteor
- Mithril
- Node.js

JavaScript is the world's most *popular* programming language.

JavaScript is the programming language of the *Web.*

JavaScript is *easy* to learn.

JavaScript is a lightweight, interpreted **programming** language.
It is designed for creating network-centric applications. It is complimentary to and integrated with Java.
JavaScript is very easy to implement because it is integrated with HTML.
It is open and cross-platform.

Hello World using Javascript

```
<html>
 <body>
 <script language = "javascript" type = "text/javascript">
       console.log("Hello World!")
</script>
</body>
</html>
```

Applications of Javascript Programming

As mentioned before, **Javascript** is one of the most widely used **programming languages** (Front-end as well as Back-end). It has it's presence in almost every area of software development. I'm going to list few of them here:

•**Client side validation** - This is really important to verify any user input before submitting it to the server and Javascript plays an important role in validting those inputs at front-end itself.

•**Manipulating HTML Pages** - Javascript helps in manipulating HTML page on the fly. This helps in adding and deleting any HTML tag very easily using javascript and modify your HTML to change its look and feel based on different devices and requirements.

•**User Notifications** - You can use Javascript to raise dynamic pop-ups on the webpages to give different types of notifications to your website visitors.

Advantages of JavaScript

The merits of using JavaScript are −

•**Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.

•**Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.

•**Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.

•**Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

•Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.

•JavaScript cannot be used for networking applications because there is no such support available.

•JavaScript doesn't have any multi-threading or multiprocessor capabilities.

JavaScript - Syntax

<script>... </script>

You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.

```
<script ...> JavaScript code </script>
```

Attributes :

•**Language** − This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.

•**Type** − This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

Comments in JavaScript

JavaScript supports both C-style and C++-style comments, Thus −

•Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.

•Any text between the characters /* and */ is treated as a comment. This may span multiple lines.

•JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.

•The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

JavaScript in <head>...</head> section

```
<html>
<head>
<script type = "text/javascript">
<!-- function sayHello() {
 alert("Hello World")
 } //-->
</script>
</head>
<body>
 <input type = "button" onclick = "sayHello()" value = "Say Hello" />
</body>
</html>
```

JavaScript in <body>...</body> section

```
<html>

<head>

</head>

<body>

 <script type = "text/javascript">

<!-- console.log("Hello World") //-->

 </script>

<p>This is web page body </p>

 </body>

</html>
```

Developer console

In the browser, users don’t see errors by default.

To see errors and get a lot of other useful information about scripts, “developer tools” have been embedded in browsers.

Google Chrome : Press `F12` The developer tools will open on the Console tab by default.

- Developer tools allow us to see errors, run commands, examine variables, and much more.
- They can be opened with `F12` for most browsers on Windows. Chrome for Mac needs `Cmd+Opt+J`, Safari: `Cmd+Opt+C` (need to enable first).

JavaScript in External File

The **script** tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

```
First.html
<html>
 <head>
<script type = "text/javascript" src = "filename.js" >
</script> </head>
 <body> ....... </body>
 </html>
```

```
Filename.js


alert("Hello World")
```

JavaScript Datatypes

JavaScript allows you to work with three primitive data types −

•**Numbers,** eg. 123, 120.50 etc.

•**Strings** of text e.g. "This text string" etc.

•**Boolean** e.g. true or false.

JavaScript also defines two trivial data types, **null** and **undefined,** each of which defines only a single value.

In addition to these primitive data types, JavaScript supports a composite data type known as **object**.

JavaScript Variables

Variables are declared with the **var** keyword as follows.

```
var money;
var name;
```

```
var money; var name;
```

JavaScript is **untyped** language.

JavaScript Variable Scope

•**Global Variables** − A global variable has global scope which means it can be defined anywhere in your JavaScript code.

•**Local Variables** − A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

```
<html>
 <body>
 <script type = "text/javascript">
var myVar = "global"; // Declare a global variable
function checkscope( ) {
 var myVar = "local"; // Declare a local variable
 console.log(myVar);
}
checkscope();
 </script>
 </body>
</html>
```

JavaScript Variable Names

•should not use any of the JavaScript reserved keywords

•should not start with a numeral (0-9). They must begin with a letter or an underscore character.

For example, **123test** is an invalid variable name but _**123test** is a valid one.

•JavaScript variable names are case-sensitive

Operator

•Arithmetic Operators

•Comparison Operators

•Logical (or Relational) Operators

•Assignment Operators

•Conditional (or ternary) Operators

Arithmetic operators

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division(/)
- Modulus(%)
- Increment (++)
- Decrement(--)
- Exponent (**)

```
let x = 5;
 let y = 3;
console.log('x + y = ', x + y);
console.log('x - y = ', x - y);
console.log('x * y = ', x * y);
console.log('x / y = ', x / y);
console.log('x % y = ', x % y);
console.log('++x = ', ++x);
console.log('x++ = ', x++);
console.log('x = ', x);
console.log('--x = ', --x);
console.log('x-- = ', x--);
console.log('x ** y =', x ** y);
```

JavaScript Comparison Operators

The JavaScript comparison operator compares the two operands.

Operator	Description	Example
==	Is equal to	10==20 : false
===	Identical (equal and of same type)	10==20 : false
!=	Not equal to	10!=20 : true
!==	Not Identical	20!==20 : false
>	Greater than	20>10 : true
>=	Greater than or equal to	20>=10 : true
<	Less than	20<10 : false
<=	Less than or equal to	20<=10 : false

```
var a = 10;
var b = 20;
result = (a == b);
console.log(result);
result = (a < b);
 console.log(result);
result = (a > b);
 console.log(result);
result = (a != b);
console.log(result);
result = (a >= b);
console.log(result);
```

JavaScript Assignment Operators

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

```
var a = 33;
var b = 10;
result = (a = b);
console.log(result);
result = (a += b);
 console.log(result);
result = (a -= b);
console.log(result);
 result = (a *= b);
console.log(result);
result = (a /= b);
 console.log(result);
result = (a %= b);
console.log(result);
```

Logical Operators

let A=true B=False

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

```
var a = true;
var b = false;
result = (a && b);
console.log(result);
result = (a || b);
console.log(result);
result = (!(a && b));
 console.log(result);
```

Type Conversions

String Conversion

String conversion happens when we need the string form of a value.

For example, `alert(value)` does it to show the value.

We can also call the `String(value)` function to convert a value to a string:

Ex:

```
let value = true;
alert(typeof value); // boolean
value = String(value); // now value is a string "true“
 alert(typeof value); // string
```

Numeric Conversion

Numeric conversion happens in mathematical functions and expressions automatically.

For example, when division `/` is applied to non-numbers:

```
alert( "6" / "2" ); // 3, strings are converted to numbers
```

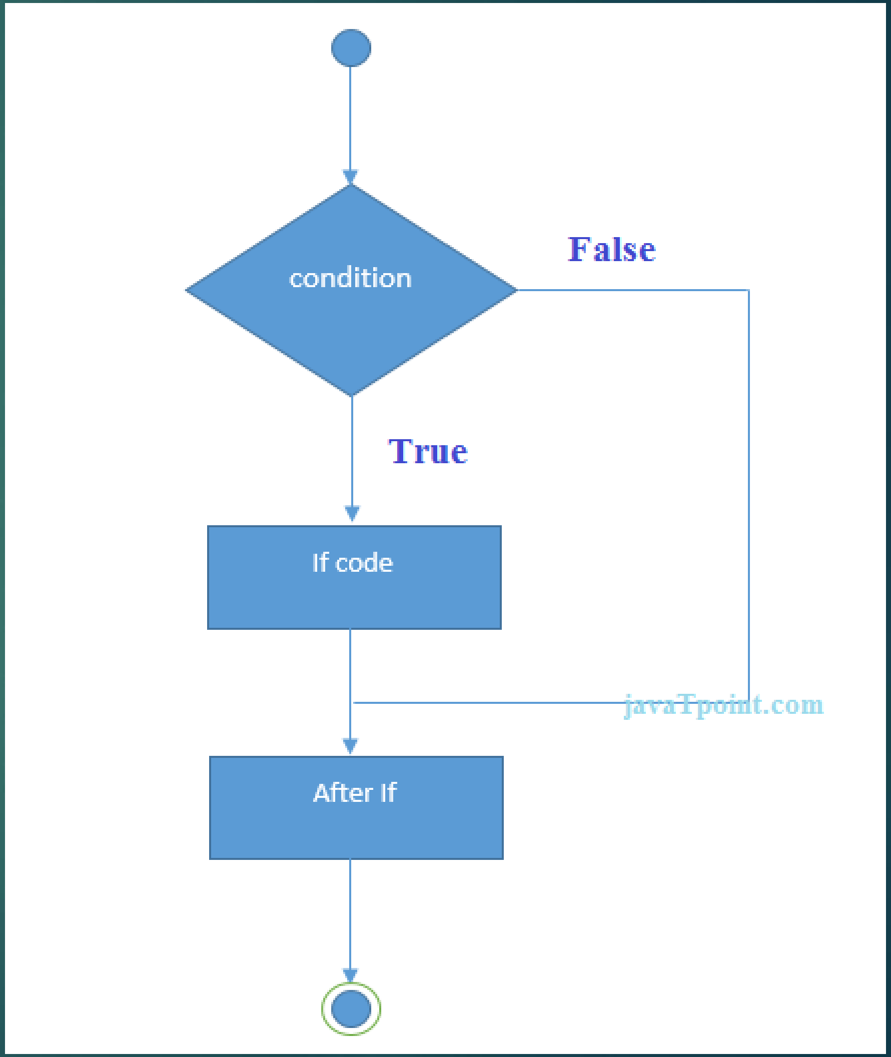
```
let str = "123";
alert(typeof str); // string
 let num = Number(str); // becomes a number 123
 alert(typeof num); // number
```

JavaScript If statement

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

```
if(expression)
{
//content to be evaluated ;
}
```

```
<script>
var a=20;
if(a>10)
{
console.log("value of a is greater than 10");
}
</script>
```



If…else Statement

It evaluates the content whether condition is true of false. The syntax of JavaScript if-e
statement is given below.

```
if(expression){
//content to be evaluated if condition is true
}
else{
//content to be evaluated if condition is false
}
```

```
<script>
var a=20;
if(a%2==0){
console.log("a is even number");
}
else{
console.log("a is odd number");
}
</script>
```

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 signature of JavaScript switch statement is given below.

```
switch(expression)
{
case value1:
 code to be executed;
 break;
case value2:
 code to be executed;
 break;
......

default:
 code to be executed if above values are not match
ed;
}
```

JavaScript Loops

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

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

A single execution of the loop body is called *an iteration*

JavaScript For loop

The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

```
for (initialization; condition; increment)

{
    code to be executed
}
```

```
<script>
for (i=1; i<=5; i++)
{
console.log(i + "<br/>")
}
</script>
```

JavaScript while loop

The **JavaScript while loop** *iterates the elements for the infinite number of times*. It should be used if number of iteration is not known. The syntax of while loop is given below.

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

```
<script>
var i=11;
while (i<=15)
{
console.log(i + "<br/>");
i++;
}
</script>
```

do while loop

The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least* once whether condition is true or false. The syntax of do while loop is given below.

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

__Example__

```
<script>
 var i=10;
 do
 {
  console.log(i);
  i--;
  }
while(i>10);
</script>
```

Breaking the loop

Normally, a loop exits when its condition becomes falsy.

But we can force the exit at any time using the special `break` directive.

```
let sum = 0;
Let i=1
 while (i<10) {
let value =5;
if (i==value) break;
 sum += i;
i++;
 } alert( 'Sum: ' + sum );
```

Continue to the next iteration

The continue directive is a "lighter version" of break.

It doesn't stop the whole loop. Instead, it stops the current iteration and forces the loop to start a new one (if the condition allows).

We can use it if we're done with the current iteration and would like to move on to the next one.

```
for (let i = 0; i < 10; i++) {
if (i % 2 == 0)
   continue;
alert(i);
}
```