Share on your Social Media

# JavaScript Tutorial for beginners

Published On: September 20, 2024

## JavaScript Tutorial for beginners

If you want to develop dynamic and interactive web pages, JavaScript is the best technology that eases the execution of complicated actions. Learn the fundamentals in this JavaScript tutorial and kickstart your web development career.

Download JavaScript Tutorial PDF

## Introduction to JavaScript

Programmers can create dynamic and interactive websites that interest viewers and perform complex tasks using JavaScript. In this JavaScript tutorial, we cover the following:

- Overview of JavaScript
- Fundamentals of JavaScript
- Functions in JavaScript
- Advantages of JavaScript

## Overview of JavaScript

JavaScript is cross-platform, lightweight, single-threaded programming. This programming language is often used to create dynamic and interactive website elements. JavaScript is a more flexible interpreted language since it executes code

line by line.

### Hello World Program in JavaScript

*console.log("Hello World!");*

"LiveScript" was the original name of JavaScript when it was first developed. JavaScript, however, has no connection to Java anymore as it developed into a completely separate language with its definition known as ECMAScript.

**JavaScript Interview Questions and Answers**

## Features of JavaScript

JavaScript is the only browser technology that does all three of the following things at once:

- Extensive HTML/CSS integration.
- Simple tasks are completed easily.
- Supported and turned on by default in all major browsers.

Join our **AngularJS course program** for a bright career in web development.

## JavaScript Code Editors

JavaScript code is written by programmers using a code editor. IDEs and lightweight editors are the two primary categories of code editors.

### IDE (Integrated Development Environment)

An extensive feature-rich editor that typically works on a "whole project" is referred to as an IDE.

- It is a complete "development environment," as the name implies, rather than merely an editor.


**MERN Stack Tutorial for Web Development Aspirants**
Published On: October 14, 2024

MERN Stack Tutorial for Web Development Aspirants There is a growing need for competent MERN...

**Tableau Developer Salary in Chennai**
Published On: October 12, 2024

Introduction A Tableau Developer designs, develops, and maintains dashboards and visualizations using Tableau software. Key...

**VMware Tutorial for Cloud Computing Aspirants**
Published On: October 12, 2024

VMware Tutorial for Cloud Computing Aspirants VMware software allows you to run a virtual machine...

- An IDE connects with a testing environment, a version management system like git, and other "*project-level*" things.
- It loads the project, which may consist of several files, permits file navigation, and offers auto-completion based on the entire project.

## IDE Options for JavaScript

- Visual Studio Code (cross-platform, free).
- WebStorm (cross-platform, paid).

Additionally, there is "Visual Studio" for Windows, which should not be confused with "Visual Studio Code."

The powerful, premium editor "Visual Studio" is only available for Windows and is ideal for the ".NET" framework. It excels at JavaScript as well. A free version of Visual Studio Community is also available.

### Lightweight Editors

"Lightweight editors" are quick, stylish, and easy to use, yet they lack the functionality of IDEs. Their primary function is to quickly open and edit files.

- An IDE functions at the project level, loading significantly more data at startup and doing necessary structural analysis of the project, etc.
- This is the primary distinction between an IDE and a "lightweight editor."
- If we only require one file, a lightweight editor operates considerably more quickly.

### Popular Choices of Lightweight Editors:

- Sublime Text (shareware, cross-platform).
- Notepad++ (free for Windows).
- Vim and Emacs for experts.

**Download JavaScript Syllabus PDF**

**VBA Macros Tutorial for Beginners**
Published On: October 10, 2024

VBA Macros Tutorial for Beginners VBA macros are programs that automate repetitive operations in Microsoft...

## Fundamentals of JavaScript

The <script> tag allows JavaScript scripts to be put virtually anywhere within an HTML text.

**Example**

```
<!DOCTYPE HTML>

<html>

<body>

 <script>

  alert( 'Hello, world!' );

 </script>

</body>

</html>
```

It is just that simple. As we see here, JavaScript code included in the <script> element is automatically run by the browser when it processes the tag.

## Variables in JavaScript

A JavaScript application typically needs to interact with data. Here are two examples:

- **Online Shop:** An online store may provide information on products for sale as well as a buying cart.
- **Chat App:** Information from a chat program could contain messages, users, and a lot more.

This information is stored in variables.

A variable is a type of data "named storage." Variables can be used to keep track of guests, gifts, and other information.

The ***let*** keyword in JavaScript is used to create variables.

The following statement declares (or creates) a variable called “message”:

*let message;*

We can now use the assignment operator =: to insert some data into it.

*let message;*

*message = ‘Hello’; // store the string ‘Hello’ in the variable named message*

At this point, the string is saved in the memory location linked to the variable. We can use the variable name to get access to it:

*let message;*

*message = ‘Hello!’;*

*alert(message); // shows the variable content*

We can condense the variable declaration and assignment into one line by doing this:

*let message = ‘Hello!’; // define the variable and assign the value*

*alert(message); // Hello!*

Additionally, we can declare many variables in a single line:

*let user = ‘John’, age = 25, message = ‘Hello’;*

Although it may seem shorter, we do not advise doing such. Please use a single line per variable to improve readability. Although a little longer, the multiline version is simpler to read:

*let user = ‘John’;*

*let age = 25;*

*let message = 'Hello';*

This multiline style is also used by some to define numerous variables:

*let user = 'John',*

*age = 25,*

*message = 'Hello';*

Alternatively, in the "comma-first" format:

*let user = 'John'*

*, age = 25*

*, message = 'Hello';*

All of these variations function in the same way, technically. Thus, everything comes down to aesthetics and personal preference.

The keywords let and var are nearly identical. It declares a variable in the same way, albeit a little more "old-fashioned."

**Javascript Developer Salary in Chennai**

## Constants in JavaScript

Use const rather than let to declare a constant or unchanging variable:

*const myBirthday = '18.04.1982';*

"Constants" are variables that are declared with const. They are not transferable. If this were attempted, an error would result:

*const myBirthday = '18.04.1982';*

*myBirthday = '01.01.2001'; // error, can't reassign the constant!*

Programmers can use the const keyword to ensure that a variable will never change and can inform all users of this fact.

## Uppercase Constants

Constants are frequently used as aliases for values that are known ahead of time but are hard to recall. These constants are named with underscores and capital letters.

Let's create constants for colors, for example, using the "web" (hexadecimal) format:

*const COLOR_RED = "#F00";*

*const COLOR_GREEN = "#0F0";*

*const COLOR_BLUE = "#00F";*

*const COLOR_ORANGE = "#FF7F00";*

*let color = COLOR_ORANGE;*

*alert(color); // #FF7F00*

## Advantages:

- "#FF7F00" is much harder to recall than COLOR_ORANGE.
- Typing "#FF7F00" incorrectly is far more common than COLOR_ORANGE.
- COLOR_ORANGE has significantly more meaning in the code than #FF7F00.

A variable is said to be "constant" if its value is constant. However, some constants like the hexadecimal representation of red are known before execution, whereas other constants are determined during runtime and remain constant after they are assigned.

## Example

*const pageLoadTime = /* time taken by a webpage to load */;*

Since pageLoadTime's value is unknown before the page loads, it is named conventionally. However, since it remains unchanged after the assignment, it remains a constant.

Put otherwise, the main purpose of capital-named constants is as aliases for "hard-coded" values.

## Rules for Declaring Variables and Constants

- Make use of names that are easy for humans to understand, such as shopping_cart or username.
- If you're not experienced, avoid using acronyms or short names like a, b, and c.
- Give names the most detail and succinctness possible. Value and data are two examples of terrible names. Names like those don't express anything.
- They should only be used if the code's context makes it very clear whatever value or data the variable is referring to.
- Decide on terms both as a team and for yourself. We should call associated variables currentUser or newUser instead of currentVisitor or newManInTown if a site visitor is referred to as a "user."

New to web development? Begin your career with our **HTML training in Chennai**.

## Types of Operators in JavaScript

In JavaScript, many types of operators are used to carry out various operations. Some of the JavaScript operators are as follows:

- Arithmetic Operator
- Comparison Operator
- Bitwise Operator
- Logical Operator
- Assignment Operator

## Arithmetic Operator

The operands are applied to arithmetic operations through the use of arithmetic operators. The following operators are referred to as JavaScript arithmetic operators:

| Operator | Description | Example |
|---|---|---|
| + | To add two operands | 12 + 12 = 24 |
| – | To subtract the second from the first operand | 42 – 10 = 32 |
| / | To divide the number by the denominator | 10/5 = 2 |
| * | To multiple two operands | 2 * 2 = 4 |
| % | To display the remainder | 5%2 = 1 |
| ++ | To increase the values by one | If a = 2, then a++ = 3 |
| — | To decrease the values by one | If a = 2, then a– = 1 |

## Comparison Operator

The two operands are compared by the JavaScript comparison operator. These are the comparison operators:

| Operator | Description | Example |
|---|---|---|
| = = | To evaluate whether or not two operands are equal. If the answer is | 7 = = 3 = false |

| | | |
|---|---|---|
| | yes, the condition is met. | |
| === | To find the identical | 1 == 2 = false |
| != | To determine whether or not two operands are equal. When the values are not equal, the condition is fulfilled. | 2!=1 = true |
| !== | The implication is that the two values are not the same. | 2!==2 = false |
| > | To determine if the left operand's value is higher than the right operand's value. | 3 > 2 = true |
| >= | To determine if the left operand's value is greater than or equal to the right operand's value. | 23 >= 23 = true |
| < | To determine if the left operand's value is smaller than | 3 < 2 = false |

| | | |
|---|---|---|
| | the right operand's value. | |
| <= | To determine if the left operand's value is less than or equal to the right operand's value. | 3 <= 3 = true |

## Bitwise Operator

Bitwise operations on operands are carried out using the bitwise operators. These bitwise operators are listed below:

| Operator | Description | Example |
|---|---|---|
| & | Each bit of its integer parameters is subjected to a boolean AND operation. | (1 == 2 & 2 == 3) = false |
| \| | On each bit of each of its integer parameters, it does a Boolean OR operation. | (1 == 1 \| 2 == 3) = true |
| ^ | Bitwise XOR action is carried out by this operator. | (1 == 2 ^ 2 == 3) = false |
| ~ | It is a unary operator that functions by flipping every | (~1) = – 1 |

| | | |
|---|---|---|
| | bit in the operand. | |
| << | The number of positions supplied in the second operand is used to shift all the bits in the first operand to the left. | (1 << 2) = 4 |
| >> | The amount of bits indicated by the right operand shifts the value of the left operand to the right. | (1>>2) = 2 |
| >>> | The bits moved in on the left of this operator are always zero, unlike the >> operator. | (1>>>2) = 2 |

## Logical Operators

All of the JavaScript logical operators are listed in the list:

| Operator | Description | Example |
|---|---|---|
| && | **Logical AND:** The condition is satisfied if both operands are non-zero. | (1 == 2 && 2 == 3) = false |
| | **Logical OR:** If any of the two | |

| | | |
|---|---|---|
| \|\| | operands is non-zero, the condition is said to be logically true. | (1 == 2 \|\| 2 == 2) = true |
| ! | **Logical Not:** reverses the operand's logical state. | !(1 == 2) = true |

## Assignment Operators

The operand can be given values by using the assignment operators. Assigning operators in JavaScript include the following:

| Operator | Description | Example |
|---|---|---|
| = | The left-side operand gets values from the right-side operand. | 2 + 1 = 3 |
| += | It assigns the outcome to the left operand after adding the right operand to the left one | a = 2; a+= 1; then, a = 3 |
| -= | It allocates the result to the left operand after subtracting the right operand from the left one | a = 3, a-=1; then, a = 2 |
| | It adds the result to the | |

| | | |
|---|---|---|
| *= | left operand after multiplying the right and left operands. | a = 1, a*= 2; then a = 2 |
| /= | The result is assigned to the left operand after the left operand and right operand are divided. | a = 4, a/=2; then a = 2 |
| %= | It performs a modulus operation with two operands and transfers the outcome to the left operand. | a = 5, a%=2; then a = 1 |

Explore a wide range of opportunities by enrolling in our **MEAN Stack course** in Chennai.

**Beginner Javascript Projects**

## Type Conversion in JavaScript

Generally in JavaScript, operators and functions automatically transform the values passed to them to the appropriate type.

For example, the *alert* automatically displays any value by converting it to a string. Values are converted to numbers using mathematical procedures.

Additionally, there are times when we must explicitly change a value to the desired type.

### String Conversion

When we require a value in its string form, string conversion takes place.

**Example:** alert(value) displays the value by doing this.

To convert a value to a string, we may alternatively use the String(value) function:

*let value = true;*

*alert(typeof value); // boolean*

*value = String(value); // now value is a string "true"*

*alert(typeof value); // string*

Most string conversions are clear. A false is changed to "false," a null to "null," and so on.

## Numeric Conversion

In mathematical expressions and functions, numerical conversion takes place automatically.

When division "/" is applied to non-numbers.

**Example:**

*alert( "6" / "2" ); // 3, the conversion of strings to numbers*

To explicitly convert a value to a number, we can use the Number(value) function as follows:

**Example**

*let str = "123";*

*alert(typeof str); // string*

*let num = Number(str); // becomes a number 123*

*alert(typeof num); // number*

When we read a value from a string-based source, such as a text form, but anticipate a number to be submitted, explicit conversion is typically necessary.

NaN is the result of such a conversion if the string is not a valid number.

*let age = Number("an arbitrary string instead of a number");*

*alert(age); // NaN, conversion failed*

## Rules for Numeric Conversion

| Value | Result |
|---|---|
| undefined | NaN |
| null | 0 |
| true and false | 1 and 0 |
| string | Whitespaces are eliminated from the beginning and finish, which includes spaces, tabs, newlines, etc. The outcome is 0 if there is an empty string left. The number is "read" from the string if not. |

**Example**

*alert( Number("  123  ") ); // 123*

*alert( Number("123z") );  // NaN (error reading a number at "z")*

*alert( Number(true) );  // 1*

*alert( Number(false) );  // 0*

Note that undefined and null behave differently in this situation: undefined becomes NaN while null becomes zero.

## Boolean Conversion
## The easiest conversion is boolean. It occurs in logical operations (condition

**tests and other such things will come up later), but it can also be done explicitly by calling Boolean(value).**

## Rules for Boolean Conversion

- Intuitively “empty” values such as 0; null; undefined; empty string; and NaN turn into false.
- Other values become true.

**Example**

*alert( Boolean(1) ); // true*

*alert( Boolean(0) ); // false*

*alert( Boolean(“hello”) ); // true*

*alert( Boolean(“”) ); // false*

## Conditional Statements in JavaScript

We have to repeat tasks frequently. JavaScript offers conditional statements like loop statements such as while and for. Using loops, you may repeat the same code repeatedly.

**Example:** Executing the same code for every number between 1 and 10 or sequentially producing items from a list.

### While Loop

The syntax for the while loop is as follows:

```
while (condition) {
  // code
  // “loop body”
}
```

The code from the loop body is run as long as the condition is true.

The loop below, for example, outputs i while i < 3:

**Example:**

```
let i = 0;

while (i < 3) { //
shows 0, then 1,
then 2

  alert( i );

  i++;

}
```

An iteration is the single execution of the loop body. In the previous example, the loop runs through three iterations.

In the previous example, if i++ was absent, the loop would (in theory) never end. In reality, the browser offers methods to break out of these loops, and server-side JavaScript allows us to end the process.

A loop condition can be any expression or variable, not only a comparison while evaluating the condition and turning it into a boolean.

For example, while (i) is a shorter form of while (i!= 0):

*let i = 3;*

*while (i) { // when i becomes 0, the condition*

*becomes false, and the loop stops*

*alert( i );*

*i–;*

*}*

### Do...While Loop

The do..while syntax can be used to shift the condition check below the body of the loop:

*do {*

*// loop body*

*} while (condition);*

The loop will run the body repeatedly while it is true, then check the condition and run the body once more.

> **Example:**
>
> *let i = 0;*
>
> *do {*
>
> *alert( i );*
>
> *i++;*
>
> *} while (i < 3);*

Only use this syntax when you want the loop's body to run at least once regardless of whether the condition is true. The alternative version is typically favored: while(...) {...}.

### For Loop

Although the for loop is the most widely used loop, it is also the most difficult. The following is the syntax.

*for (begin; condition; step) {*

*// ... loop body .. }*

Let's use these components as examples to understand their meaning. The loop that follows executes alert(i) for each i between 0 and (but not including) 3:

*for (let i = 0; i < 3; i++) { // shows 0, then 1, then 2*

*alert(i);*

*}*

### General Loop Algorithm

*Run begin*

*→ (if condition → run body and run step)*

*→ (if condition → run body and run step)*

*→ (if condition → run body and run step)*

*→ ...*

**Example:**

*// for (let i = 0; i < 3; i++) alert(i)*

*// run begin*

*let i = 0*

```
// if condition → run body and run step

if (i < 3) { alert(i); i++ }

// if condition → run body and run step

if (i < 3) { alert(i); i++ }

// if condition → run body and run step

if (i < 3) { alert(i); i++ }

// ...finish, because now i == 3
```

### Breaks in Loop

A loop usually breaks when its condition is no longer true. However, we can use the specific break directive to force the exit at any time.

The loop below, for instance, requests a string of numbers from the user and "breaks" if no number is entered:

```
let sum = 0;

while (true) {

  let value = +prompt("Enter a number", '');

  if (!value) break; // (*)

  sum += value;

}

alert( 'Sum: ' + sum );
```

If the user cancels the input or enters an empty line, the break directive is triggered at line (*). It immediately ends the loop and moves control to the line that comes after it. Specifically, be mindful.

When a loop's condition needs to be verified somewhere along its body, rather than at its start or finish, the "infinite loop + break as needed" combo works incredibly well.

### Continue in Loop

The command "lighter version" of break is "continue." The entire cycle is not stopped. Rather, it compels the loop to begin a new iteration (if the condition permits) and ends the present one.

If we're finished with this iteration and want to go on to the next, we can use it. The loop below keeps producing only odd values.

```
for (let i = 0; i < 10; i++) {

  if (i % 2 == 0) continue;

  alert(i); // 1, then 3, 5, 7, 9   }
```

The continue directive ends the body's execution for even values of i and transfers control to the next for loop (with the next number). As a result, only odd values trigger the alarm.

Learn the popular JavaScript framework with the **React.JS course** to design impressive websites.

## Switch Statement in JavaScript

Multiple if checks can be replaced by a switch statement. It provides a more illustrative method of comparing a value with several variations. The switch has an optional default and one or more case blocks.

**Syntax:**

```
switch(x) {

  case 'value1':  // if (x === 'value1')

    ...

    [break]
```

```
  case 'value2':  // if (x === 'value2')
    ...
    [break]
  default:
    ...
    [break]
}
```

- Strict equality with the value from the first case (value 1), the value from the second (value 2), and so forth, is checked for the value of x.
- The switch begins running the code from the matching case until the closest break (or until the switch ends) if equality is discovered.
- If no case matches, the default code—if any—is run.

**Example**

```
let a = 2 + 2;
switch (a) {
  case 3:
    alert( 'Too small' );
    break;
  case 4:
    alert( 'Exactly!' );
    break;
  case 5:
    alert( 'Too big' );
    break;
  default:
```

*alert( "I don't know such values" );  }*

At this point, the switch begins comparing a with the first case variant, which is 3. The game is lost. Next, 4. Given that there is a match, case 4 is executed up until the closest break.

## Grouping of Case

Case variations that have the same code might be grouped.

For example, if we wish case 3 and case 5 to use the same code:

```
let a = 3;

switch (a) {

  case 4:

    alert('Right!');

    break;

  case 3: // (*) grouped two cases

  case 5:

    alert('Wrong!');

    alert("Why don't you take a math class?");

    break;

  default:

    alert('The result is strange. Really.');

}
```

At this point, 3 and 5 display the same message. The ability to "group" situations is a side consequence of how the switch/case operates without a break. Because there is no break, case 3 execution in this instance begins at line (*) and continues through case 5.

## Type Cases

The equality check is never lenient. For the values to match, they must be of the same type.

```
let arg = prompt("Enter a value?");

switch (arg) {

  case '0':

  case '1':

    alert( 'One or zero' );

    break;

  case '2':

    alert( 'Two' );

    break;

  case 3:

    alert( 'Never executes!' );

    break;

  default:

    alert( 'An unknown value' );  }
```

- The first alert sounds for 0 and 1.
- The second alert goes off for 2.
- The question yields a string "3" for the number 3, yet it is not exactly equal to 3. Thus, case 3 has a dead code! It will run the default variant.

Our **Python web development course** will be helpful for you to accelerate your career.

**Java Script Code Challenges**

## Functions in JavaScript

We frequently have to carry out a comparable

action throughout the script.

For example, we must display a visually appealing message when a visitor checks in, logs out, and possibly even someplace else.

The primary "building blocks" of the program are its functions. They permit the code to be called repeatedly without becoming repetitive.

## Function Declaration

A function declaration can be used to create a function.

### Syntax

```
function showMessage() {

  alert( 'Hello everyone!' );

}
```

The function keyword appears first, followed by the function's name, a list of parameters, and finally the function's code, also known as "the function body," enclosed in curly brackets.

```
function name(parameter1, parameter2, ...
parameterN) {

 // body    }
```

**Example**

```
function showMessage() {

  alert( 'Hello everyone!' );

}

showMessage();

showMessage();
```

The function's code is run by the call showMessage(). The message will appear twice in this instance. The avoidance of code duplication is

a primary goal of functions, as this example amply illustrates.

It only takes editing the code in the function that outputs the message if we ever need to change the message or how it is shown.

## Local Variables

Only the function in which it is declared can see the variable.

**Example**

```
function showMessage() {

  let message = "Hello, I'm JavaScript!"; // local variable

  alert( message );

}

showMessage(); // Hello, I'm JavaScript!

alert( message ); // <– Error! The variable is local to the function.
```

## Outer Variables

An external variable can also be accessed by a function.

**Example:**

```
let userName = 'John';

function showMessage() {

  let message = 'Hello, ' + userName;

  alert(message);

}

showMessage(); // Hello, John
```

The outer variable is fully accessible to the function.

It is also editable.

**Example**

```
let userName = 'John';

function showMessage() {

  userName = "Bob"; // (1) changed the outer variable

  let message = 'Hello, ' + userName;

  alert(message);

}

alert( userName ); // John before the function call

showMessage();

alert( userName ); // Bob, the value was modified by the function
```

Only in the absence of a local variable is the outer variable used.

A variable with the same name that is declared inside a function shadows the external one.

**Example**

```
let userName = 'John';

function showMessage() {

  let userName = "Bob"; //local variable

  let message = 'Hello, ' + userName;

  alert(message);    }

showMessage();

alert( userName ); // The code did not visit the outer variable, leaving John unaltered.
```

## Parameters

With arguments, we may give functions any kind of data.

**Example**

```
function showMessage(from, text) {

  alert(from + ': ' + text);

}

showMessage('Ann', 'Hello!');

showMessage('Ann', "What's up?");
```

The supplied values are copied to local variables from and text when the function is called in lines (*) and (**). The function then employs them.

A function always receives a copy of the value, the change is not visible from the outside:

**Example:**

```
function showMessage(from, text) {

  from = '*' + from + '*';

  alert( from + ': ' + text );

}

let from = "Ann";

showMessage(from, "Hello"); // *Ann*: Hello

alert( from );
```

A value is also referred to as an argument when it is supplied as a function parameter.

- The variable (a declaration time term) included in parenthesis in the function declaration is called a parameter.
- The value that is supplied to the function when it is called—a call time term—is known as an argument.

Functions are declared with a list of their parameters, and then we call them with arguments.

Enhance your skills to become a **full-stack developer** with our exclusive training.

## Default Values

The associated value becomes undefined when a function is invoked without an argument.

**Example:** The method showMessage(from, text) stated before can be invoked with just one argument:

*showMessage("Ann");*

That isn't an error. The result of such a call would be "*Ann*: undefined." Since the text's value isn't passed, it turns undefined.

In the function declaration, we can use =: to set the so-called "default" (to use if omitted) value for a parameter.

```
function showMessage(from, text = "no text given")
{

  alert( from + ": " + text );

}

showMessage("Ann");
```

If the argument is present, the default value also enters but strictly equals undefined, like in this case:

*showMessage("Ann", undefined);*

In this case, "no text given" is a string, but it might also be a more intricate expression that is only assessed and allocated if a parameter is not there.

```
function showMessage(from, text =
anotherFunction()) {

  // anotherFunction() only executed if no text given
```

```
  // its result becomes the value of text
}
```

## Returning Value

As a result, a function can return a value to the caller code.

A function that adds two values would be the most straightforward example:

```
function sum(a, b) {
  return a + b;
}
let result = sum(1, 2);
alert( result ); // 3
```

### Rules for naming functions:

- A function's name should make it obvious what it does.
- Since a function is an action, its names are typically spoken.
- Numerous popular function prefixes, such as create, display, get, verify, and so forth, are available. They can be used to suggest what a function accomplishes.

**Java Script Online Training**

## Advantages of JavaScript

Following are the advantages of learning JavaScript for your web development career.

- **Flexibility:** JavaScript may be utilized to create mobile apps, games, websites, and more.
- **Client and Server-Side:** JavaScript is currently utilized for server-side application

development due to frameworks like Node.js and Express.js.

- **End-to-End Solutions:** JavaScript gives programmers the ability to write comprehensive fixes for a range of issues.
- **Continuous Evolution:** New features and standards are always being added to JavaScript.
- **Bright Community:** A sizable user and mentor community actively supports the development of JavaScript.

## Conclusion

We have covered the fundamental concepts in this JavaScript tutorial. Explore more in our **JavaScript training in Chennai** to start a promising career in web development.

Share on your Social Media


**Softlogic Academy**

## Softlogic Systems

**KK Nagar [Corporate Office]**

No.10, PT Rajan Salai, K.K. Nagar, Chennai – 600 078.
**Landmark:** Karnataka Bank Building
**Phone:** +91 86818 84318
**Email:** enquiry@softlogicsys.in
**Map:** Google Maps Link

**OMR**

No. E1-A10, RTS Food Street
92, Rajiv Gandhi Salai (OMR),
Navalur, Chennai - 600 130.

**Landmark:** Adj. to AGS Cinemas

**Phone:** +91 89256 88858

**Email:** info@softlogicsys.in

**Map:** Google Maps Link

Disclaimer

Privacy Policy

Terms and Conditions

## Courses

- Python
- Software Testing
- Full Stack Developer
- Java
- Power BI
- Clinical SAS
- Data Science
- Embedded
- Cloud Computing
- Hardware and Networking
- VBA Macros
- Mobile App Development
- DevOps

## Social Media Links

## Review Sources

- Google
- Trustpilot
- Glassdoor
- Mouthshut
- Sulekha
- Justdial
- Ambitionbox
- Indeed
- Software Suggest
- Sitejabber