Unit-3 Java Script

Introduction

JavaScript (JS) is the most popular lightweight, interpreted compiled programming language. It can be used for both **Client-side** as well as **Server-side** developments. JavaScript also known as a scripting language for web pages.

Reason to Learn JavaScript

JavaScript is used by many developers (65% of the total development community), and the number is increasing day by day. JavaScript is one such programming language that has more than 1444231 libraries and increasing rapidly. It is preferred over any other programming language by most developers. Also, major tech companies like Microsoft, Uber, Google, Netflix, and Meta use JavaScript in their projects.

JavaScript can be added to your HTML file in two ways:

- Internal JavaScript
- External JavaScript

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

Example: It is the basic example of using JavaScript code inside of HTML code, that script enclosing section can be placed in the body or head of the HTML document.

HTML

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>
        Basic Example to Describe JavaScript
    </title>
</head>

<body>
```

```
    <!-- JavaScript Code -->
    <script>
        console.log("Welcome to MCA");
    </script>
</body>


</html>
```

Run on IDE

Output: The output will display on console.

```
Welcome to MCA
```

External JavaScript: We can create the file with a .js extension and paste the JS code inside of it. After creating the file, add this file in <script src="file_name.js"> tag, and this <sctipt> can import inside <head> or <body> tag of the HTML file.

Example: It is the basic example of using JavaScript javascript code which is written in a different file. By importing that .js file in the head section.

HTML

```
<!DOCTYPE html>
<html lang="en">


<head>
    <title>
        Basic Example to Describe JavaScript
    </title>
    <script src="main.js"></script>
</head>


<body>
</body>


</html>
```

Javascript

```
/* JavaScript code can be embedded inside
head section or body section */
console.log("Welcome to MCA");

// JavaScript file name: main.js
```

Output: The output will display on console.

```
Welcome to MCA
```

JavaScript Used for

It is mainly used to develop websites and web-based applications. JavaScript is a language that can be used as a front-end as well as a backend.

- **Creating Interactive Websites:** JavaScript is used to make web pages dynamic and interactive. It means using JavaScript, we can change the web page content and styles dynamically.
- **Building Applications:** JavaScript is used to make web and mobile applications. To build web and mobile apps, we can use the most popular JavaScript frameworks like – ReactJS, React Native, Node.js etc.
- **Web Servers:** We can make robust server applications using JavaScript. To be precise we use JavaScript frameworks like Node.js and Express.js to build these servers.
- **Game Development:** JavaSCript can be used to design Browser games. In JavaScript, lots of game engines are available that provide frameworks for building games.

How JavaScript makes HTML build-website better

- JavaScript is an advanced programming language that makes web pages more interactive and dynamic whereas HTML is a standard markup language that provides the primary structure of a website.
- JavaScript simply adds dynamic content to websites to make them look good and HTML work on the look of the website without the interactive effects and all.
- JavaScript manipulates the content to create dynamic web pages whereas HTML pages are static which means the content cannot be changed.
- JavaScript is not cross-browser compatible whereas HTML is cross-browser compatible.
- JavaScript can be embedded inside HTML but HTML can not be embedded inside JavaScript.

Things that makes JavaScript demanding

JavaScript is the most popular and hence the most loved language around the globe. Apart from this, there are abundant reasons to become the most demanding. Below are a listing of a few important points:

- **No need for compilers:** Since JavaScript is an interpreted language, therefore it does not need any compiler for compilation.
- **Used both Client and Server-side:** Earlier JavaScript was used to build client-side applications only, but with the evolution of its frameworks namely Node.js and Express.js, it is now widely used for building server-side applications too.
- **Helps to build a complete solution:** As we saw, JavaScript is widely used in both client and server-side applications, therefore it helps us to build an end-to-end solution to a given problem.
- **Used everywhere:** JavaScript is so loved because it can be used anywhere. It can be used to develop websites, games or mobile apps, etc.
- **Huge community support:** JavaScript has a huge community of users and mentors who love this language and take it's legacy forward.

Difference Between JavaScript and Java

JavaScript	Java
Object-oriented. No distinction between types of objects. Inheritance is through the prototype mechanism, and properties and methods can be added to any object dynamically.	Class-based. Objects are divided into classes and instances with all inheritance through the class hierarchy. Classes and instances cannot have properties or methods added dynamically.
Variable data types are not declared (dynamic typing).	Variable data types must be declared (static typing).
Cannot automatically write to hard disk.	Cannot automatically write to hard disk.

History of JavaScript: It was created in 1995 by Brendan Eich while he was an engineer at Netscape. It was originally going to be named LiveScript but was renamed. Unlike most programming languages, the JavaScript language has no concept of input or output. It is designed to run as a scripting language in a host environment, and it is up to the host environment to provide mechanisms for communicating with the outside world. The most common host environment is the browser.

Features of JavaScript: According to a recent survey conducted by **Stack Overflow**, JavaScript is the most popular language on earth.
With advances in browser technology and JavaScript having moved into the server with Node.js and other frameworks, JavaScript is capable of so much more. Here are a few things that we can do with JavaScript:

- JavaScript was created in the first place for DOM manipulation. Earlier websites were mostly static, after JS was created dynamic Web sites were made.
- Functions in JS are objects. They may have properties and methods just like another object. They can be passed as arguments in other functions.

- Can handle date and time.
- Performs Form Validation although the forms are created using HTML.
- No compiler is needed.

Applications of JavaScript:

- **Web Development:** Adding interactivity and behavior to static sites JavaScript was invented to do this in 1995. By using AngularJS that can be achieved so easily.
- **Web Applications:** With technology, browsers have improved to the extent that a language was required to create robust web applications. When we explore a map in Google Maps then we only need to click and drag the mouse. All detailed view is just a click away, and this is possible only because of JavaScript. It uses Application Programming Interfaces(APIs) that provide extra power to the code. The Electron and React is helpful in this department.
- **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.
- **Games:** Not only in websites, but JavaScript also helps in creating games for leisure. The combination of JavaScript and HTML 5 makes JavaScript popular in game development as well. It provides the EaseJS library which provides solutions for working with rich graphics.
- **Smartwatches:** JavaScript is being used in all possible devices and applications. It provides a library PebbleJS which is used in smartwatch applications. This framework works for applications that require the internet for its functioning.
- **Art:** Artists and designers can create whatever they want using JavaScript to draw on HTML 5 canvas, and make the sound more effective also can be used **p5.js** library.
- **Machine Learning:** This JavaScript ml5.js library can be used in web development by using machine learning.
- **Mobile Applications:** JavaScript can also be used to build an application for non-web contexts. The features and uses of JavaScript make it a powerful tool for creating mobile applications. This is a Framework for building web and mobile apps using JavaScript. Using React Native, we can build mobile applications for different operating systems. We do not require to write code for different systems. Write once use it anywhere!

Limitations of JavaScript:

- **Security risks:** JavaScript can be used to fetch data using AJAX or by manipulating tags that load data such as <img>, <object>, <script>. These attacks are called cross site script attacks. They inject JS that is not the part of the site into the visitor's browser thus fetching the details.
- **Performance:** JavaScript does not provide the same level of performance as offered by many traditional languages as a complex program written in JavaScript would be comparatively slow. But as JavaScript is used to perform simple tasks in a browser, so performance is not considered a big restriction in its use.
- **Complexity:** To master a scripting language, programmers must have a thorough knowledge of all the programming concepts, core language objects, client and server-side objects otherwise it would be difficult for them to write advanced scripts using JavaScript.

- **Weak error handling and type checking facilities:** It is weakly typed language as there is no need to specify the data type of the variable. So wrong type checking is not performed by compile.

Why JavaScript is known as a lightweight programming language?

JavaScript is considered as lightweight due to the fact that it has low CPU usage, is easy to implement and has a minimalist syntax. Minimalist syntax as in, it has no data types. Everything is treated here as an object. It is very easy to learn because of its syntax similar to C++ and Java.

A lightweight language does not consume much of your CPU's resources. It doesn't put excess strain on your CPU or RAM. JavaScript runs in the browser even though it has complex paradigms and logic which means it uses fewer resources than other languages. For example, NodeJs, a variation of JavaScript not only performs faster computations but also uses less resources than its counterparts such as Dart or Java.

Additionally, when compared with other programming languages, it has less in-built libraries or frameworks, contributing as another reason for it to be lightweight. However, this brings it a drawback that we need to incorporate external libraries and frameworks.

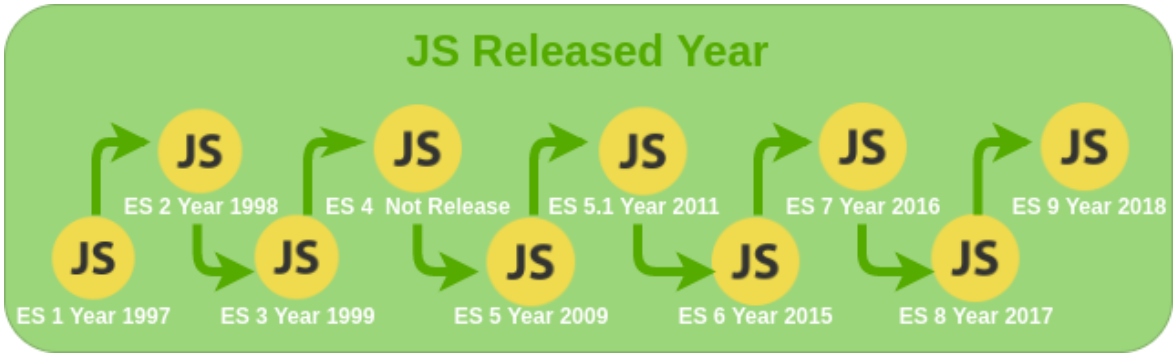
Is JavaScript compiled or interpreted or both?

JavaScript is both compiled and interpreted. In the earlier versions of JavaScript, it used only the interpreter that executed code line by line and shows the result immediately. But with time the performance became an issue as interpretation is quite slow. Therefore, in the newer versions of JS, probably after the V8, JIT compiler was also incorporated to optimize the execution and display the result more quickly. This JIT compiler generates a bytecode that is relatively easier to code. This bytecode is a set of highly optimized instructions. The V8 engine initially uses an interpreter, to interpret the code. On further executions, the V8 engine finds patterns such as frequently executed functions, frequently used variables, and compiles them to improve performance.

JavaScript is best known for web page development but it is also used in a variety of non-browser environments. You can learn JavaScript from the ground up by following this JavaScript Tutorial and JavaScript Examples.
JS stands for **JavaScript**. It is a lightweight, cross-platform, and interpreted scripting language. It is well-known for the development of web pages, many non-browser environments also use it. JavaScript can be used for Client-side developments as well as Server-side developments. JavaScript contains a standard library of objects, like Array, Date, and Math, and a core set of language elements like operators, control structures, and statements. You can check the

Introduction to JavaScript.

JS version release year:



Characteristics of JS:

- **Platform independent:** JavaScript runs on browsers which is available on all kinds of machines and is hence platform independent.
- **Dynamically typed languages:** This language can receive different data types over time.
- **Case Sensitive Format:** JavaScript is case sensitive so you have to be aware of that.
- **Light Weight:** It is so lightweight, and all the browsers are supported by JS.
- **Handling:** Handling events is the main feature of JS, it can easily respond on the website when the user tries to perform any operation.
- **Interpreter Centered:** JavaScript is built with interpreter centered that allows the user to get the output without the use of the compiler.

Advantages of JS:

- JavaScript is executed on the user's browsers not on the webserver so it saves bandwidth and load on the webserver.
- The JavaScript language is easy to learn it offers syntax that is close to English language.
- In JavaScript, if you ever need any certain feature then you can write it by yourself and use an add-on like **Greasemonkey** to implement it on the web page.
- It does not require a compilation process so no compiler is needed user's browsers do the task.
- JavaScript is easy to debug, and there are lots of frameworks available that you can use and become master on that.
- It is constantly being improved and newer features are being introduced that significantly tamp down the lines of code for web applications. For example, Arrow Functions were introduced in the ES6 version which provides a short syntax to write anonymous functions.

Disadvantages of JS:

- JavaScript codes are visible to the user so user can place some code into the site that compromises the security of data over the website. That will be security issue.
- All browsers interpret JavaScript that is correct, but they interpret it differently from each other.
- It only supports single inheritance, so in few cases may require the object-oriented language characteristic.
- A single error in code can totally stop the website's code rendering on the website.
- JavaScript stores numbers as 64-bit floating-point numbers but operators operate on 32-bit bitwise operands. JavaScript first converts the number to 32 bit, performs the operation, and converts it to 64 bit again which reduces its speed.

1. Client-side scripting :

Web browsers execute client-side scripting. It is used when browsers have all code. Source code is used to transfer from webserver to user's computer over the internet and run directly on browsers. It is also used for validations and functionality for user events.
It allows for more interactivity. It usually performs several actions without going to the user. It cannot be basically used to connect to databases on a web server. These scripts cannot access the file system that resides in the web browser. Pages are altered on basis of the user's choice. It can also be used to create "cookies" that store data on the user's computer.

2. Server-side scripting :

Web servers are used to execute server-side scripting. They are basically used to create dynamic pages. It can also access the file system residing at the webserver. A server-side environment that runs on a scripting language is a web server.
Scripts can be written in any of a number of server-side scripting languages available. It is used to retrieve and generate content for dynamic pages. It is used to require to download plugins. In this load times are generally faster than client-side scripting. When you need to store and retrieve information a database will be used to contain data. It can use huge resources of the server. It reduces client-side computation overhead. The server sends pages to the request of the user/client.

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

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.

Client-side scripting	Server-side scripting
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.
It reduces load on processing unit of the server.	It surge the processing load on the server.

Client-side scripting

Client-side scripting is when the server sends the code along with the HTML web page to the client. The script is referred to by the code.

In other words, client-side scripting is a method for browsers to run scripts without having to connect to a server.

The code runs on the client's computer's browser either while the web page is loading or after it has finished loading.

Client-side scripting is mostly used for dynamic user interface components including pull-down menus, navigation tools, animation buttons, and data validation.

It is currently quickly expanding and evolving on a daily basis. As a result, creating client-side web programming has become easier and faster, lowering server demand.

By far the most popular client-side scripting languages or web scripting languages, JavaScript and jQuery are frequently utilized to construct dynamic and responsive webpages and websites.

The browser downloads the code to the local machine (temporarily) and begins processing it without the server. As a result, client-side scripting is browser-specific.

What is Client-Side Script, and how does it work?

A client-side script is a tiny program (or collection of instructions) that is put into a web page. It is handled by the client browser rather than the web server.

The client-side script, along with the HTML web page it is embedded in, is downloaded from the server at the client end. The code is interpreted and executed by the web browser, which then displays the results on the screen.

The client refers to the script that runs on the user's computer system. It can either be integrated (or injected) into the HTML content or stored in a separate file (known as an external script).

When the script files are requested, they are transmitted from the web server (or servers) to the client system. The script is run by the client's web browser, which subsequently displays the web page, including any visible script output.

Client-side scripts may also include instructions for the web browser to follow in response to user activities like clicking a page button. If a client wants to see the source code of a web page, they can typically look at them.

Client-side Scripting Languages are widely used.

Client-side scripting language or client-side programming refers to a language in which a client-side script or program is written utilizing syntax.

The following are the most popular client-side scripting languages:

1. **JavaScript** is the most commonly used client-side scripting or programming language. It is written in the ECMAScript programming language.

JavaScript is a dynamically typed (also known as weakly typed) object-oriented scripting language. It uses an integrated interpreter to run directly in the browser.

Weakly typed indicates that the variables can be implicitly transformed from one data type to another.

2. **VBScript**: This scripting language was developed by Microsoft, based on Visual Basic. It was mostly used to improve the functionality of web pages in Internet Explorer. The Internet Explorer web browser interprets VBScript. No one really uses it now.

3. **jQuery**: jQuery is a JavaScript library that is fast, tiny, and lightweight. It's used to turn a lot of JavaScript code into user-friendly functionality.

The jQuery language is used by the majority of the world's largest firms, including Google, Microsoft, IBM, Netflix, and others.

Scripting on the client-side

Client-side scripting is a technique for enhancing the interactivity of online pages or websites. It's mostly utilized on the front end, where the user can see what's going on through their browser. The following are some of the most common uses of client-side scripting:

- To get data from a user's screen or a web browser.
- In the realm of online games, this term is used.
- To make changes to a web page without having to reload it.

Validation is done using client-side scripting. If a user enters invalid credentials on the login page, the web page displays an error notice on the client machine instead of sending the information to the web server.

- Instead of simply displaying visuals, design ad banners that interact with the user.
- To make animated pictures that change as the mouse moves over them.
- A client-side script can be used to identify installed plug-ins and alert the user if one is needed.

The Benefits of Client-Side Scripting

The following are some of the many benefits of client-side scripting:

1. Client-side scripting is a simple language to learn and utilize. It only necessitates rudimentary programming knowledge or experience.
2. Client-side scripting has the advantage of being lightweight and reasonably simple to implement (syntax not too complex). The code modification and execution are both quick.
3. Data processing is done on the client side instead of the server, making large-scale applications easier to scale. As a result, the server's burden is reduced.
4. Data validation on the client side can be accomplished using a client-side scripting language such as JavaScript.
5. Client-side script execution is faster since the script is downloaded from the server and executed directly on the user's machine via the browser.
6. Client-side scripting can also be used for mathematical assessment.
7. Client-side programming facilitates the completion of complex activities in a limited number of steps.
8. Script code that is only performed by the browser and not by the server.
9. Executing script code takes far too little time.
10. When a user taps a key, moves the mouse, or clicks, the browser responds promptly.

Client-side Scripting's Drawbacks

The following are some of the disadvantages of client-side scripting:

1. Client-side scripting is insecure since the code is transmitted to the client as is, and therefore visible to it if the client looks at the source code of his web page. In a nutshell, code is almost always visible.
2. If we need to access databases or send sensitive data over the internet, client-side programming is not an option.
3. There is no guarantee that the user's browser has JavaScript enabled. As a result, notwithstanding the possibility of offloading, all essential functions must be loaded on the server.
4. The script's (or program's) smooth operation is entirely dependent on the client's browser, its settings, and its security level.
5. Debugging and maintaining a web application relying on excessive JavaScript might be difficult.
6. Client-side scripting languages are typically more constrained than server-side scripting languages in terms of capabilities.

JavaScript Output

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>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

Changing the innerHTML property of an HTML element is a common way to display data in HTML.

Using document.write()

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

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

Using document.write() after an HTML document is loaded, will delete all existing HTML:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```

The document.write() method should only be used for testing.

Using window.alert()

You can use an alert box to display data:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
```

```
</script>

</body>
</html>
```
You can skip the window keyword.

In JavaScript, the window object is the global scope object. This means that variables, properties, and methods by default belong to the window object. This also means that specifying the window keyword is optional:

Example
```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
alert(5 + 6);
</script>

</body>
</html>
```
Using console.log()
For debugging purposes, you can call the console.log() method in the browser to display data.

You will learn more about debugging in a later chapter.

Example
```
<!DOCTYPE html>
<html>
<body>

<script>
console.log(5 + 6);
</script>
</body>
</html>
```
JavaScript Print
JavaScript does not have any print object or print methods.

You cannot access output devices from JavaScript.

The only exception is that you can call the window.print() method in the browser to print the content of the current window.

Example
```
<!DOCTYPE html>
```

```
<html>
<body>

<button onclick="window.print()">Print this page</button>

</body>
</html>
```

JavaScript Data Types

JavaScript has 8 Datatypes

1. String

2. Number

3. Bigint

4. Boolean

5. Undefined

6. Null

7. Symbol

8. Object

The Object Datatype

The object data type can contain:

1. An object

2. An array

3. A date

```
// Numbers:
let length = 16;
let weight = 7.5;

// Strings:
let color = "Yellow";
let lastName = "Johnson";

// Booleans
let x = true;
```

```
let y = false;

// Object:
const person = {firstName:"John", lastName:"Doe"};

// Array object:
const cars = ["Saab", "Volvo", "BMW"];

// Date object:
const date = new Date("2022-03-25");
```

JavaScript Variables

What are Variables?

Variables are containers for storing data (storing data values).

4 Ways to Declare a JavaScript Variable:

- Using `var`
- Using `let`
- Using `const`
- Using nothing

Things To Remember While Naming A Variable

- Variable names should be short and easy to remember.
- They should be descriptive enough to tell you what the variable represents.
- They should not be too generic or too specific to avoid confusion.
- They should not include any special characters like $, %, or @, except underscore.
- They should not contain spaces.
- They should not start with a number.
- Start a variable name with a letter or underscore(_).
- Javascript variables are case-sensitive, i.e., x is not equal to X.
- They should be unique and not used by other variables in the code.

- They could either be camel-cased or lowercase.
- They should not start with a capital letter.

There are three types of keywords for a variable declaration in JS: var, let, and const.

Example

```
var x;
const y;
let z;
```

1. **Var -** It is an outdated keyword in Javascript. We cannot declare a variable without var in older browsers. The variable declared using var can store variable values during the program's execution and can be redeclared anytime.
2. **Const -** It is a keyword used to declare constant variables whose value cannot be changed throughout the program.
3. **Let -** It is a newer keyword for declaring variables whose values can be changed. In this, a variable cannot be declared multiple times; however, they are declared once in a program.

In this example, `x`, `y`, and `z`, are variables, declared with the `var` keyword:

Example

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

Try it Yourself »

In this example, `x`, `y`, and `z`, are variables, declared with the `let` keyword:

Example

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

Try it Yourself »

In this example, `x`, `y`, and `z`, are undeclared variables:

Example

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

Try it Yourself »

From all the examples above, you can guess:

- x stores the value 5
- y stores the value 6
- z stores the value 11

When to Use JavaScript var?

Always declare JavaScript variables with `var`,`let`, or`const`.

The `var` keyword is used in all JavaScript code from 1995 to 2015.

The `let` and `const` keywords were added to JavaScript in 2015.

If you want your code to run in older browsers, you must use `var`.

When to Use JavaScript const?

If you want a general rule: always declare variables with `const`.

If you think the value of the variable can change, use `let`.

In this example, `price1`, `price2`, and `total`, are variables:

Example

```
const price1 = 5;
const price2 = 6;
```

```
let total = price1 + price2;
```

Try it Yourself »

The two variables `price1` and `price2` are declared with the `const` keyword.

These are constant values and cannot be changed.

The variable `total` is declared with the `let` keyword.

This is a value that can be changed.

JavaScript Let

❮ PreviousNext ❯

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` cannot be **redeclared**.

You cannot accidentally redeclare a variable.

With `let` you can not do this:

Example

```
let x = "John Doe";

let x = 0;

// SyntaxError: 'x' has already been declared
```

With `var` you can:

Example

```
var x = "John Doe";

var x = 0;
```

Block Scope

Before ES6 (2015), JavaScript had only **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
```

Try it Yourself »

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
```

Try it Yourself »

JavaScript Const

❮ PreviousNext ❯

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
```

Try it Yourself »

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;
```

When to use JavaScript const?

Always declare a variable with `const` when you know that the value should not be changed.

Use `const` when you declare:

- A new Array
- A new Object
- A new Function
- A new RegExp

Constant Objects and Arrays

The keyword `const` is a little misleading.

It does not define a constant value. It defines a constant reference to a value.

Because of this you can NOT:

- Reassign a constant value
- Reassign a constant array
- Reassign a constant object

But you CAN:

- Change the elements of constant array
- Change the properties of constant object

Constant Arrays

You can change the elements of a constant array:

Example

```
// You can create a constant array:
const cars = ["Saab", "Volvo", "BMW"];

// You can change an element:
cars[0] = "Toyota";

// You can add an element:
cars.push("Audi");
```

Try it Yourself »

But you can NOT reassign the array:

Example

```
const cars = ["Saab", "Volvo", "BMW"];

cars = ["Toyota", "Volvo", "Audi"];    // ERROR
```

Try it Yourself »

Constant Objects

You can change the properties of a constant object:

Example

```
// You can create a const object:
const car = {type:"Fiat", model:"500", color:"white"};

// You can change a property:
car.color = "red";

// You can add a property:
car.owner = "Johnson";
```

Try it Yourself »

But you can NOT reassign the object:

Example

```
const car = {type:"Fiat", model:"500", color:"white"};

car = {type:"Volvo", model:"EX60", color:"red"};    // ERROR
```

Try it Yourself »

JavaScript Data Types

❮ PreviousNext ❯

JavaScript has 8 Datatypes

1. String
2. Number
3. Bigint
4. Boolean
5. Undefined
6. Null
7. Symbol
8. Object

The Object Datatype

The object data type can contain:

1. An object
2. An array
3. A date

Examples

```
// Numbers:
let length = 16;
let weight = 7.5;

// Strings:
let color = "Yellow";
let lastName = "Johnson";

// Booleans
let x = true;
let y = false;

// Object:
const person = {firstName:"John", lastName:"Doe"};

// Array object:
const cars = ["Saab", "Volvo", "BMW"];

// Date object:
const date = new Date("2022-03-25");
```

Note

A JavaScript variable can hold any type of data.

The Concept of Data Types

In programming, data types is an important concept.

To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

```
let x = 16 + "Volvo";
```

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

```
let x = "16" + "Volvo";
```

Note

When adding a number and a string, JavaScript will treat the number as a string.

Example

```
let x = 16 + "Volvo";
```

Try it Yourself »

Example

```
let x = "Volvo" + 16;
```

Try it Yourself »

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

JavaScript:

```
let x = 16 + 4 + "Volvo";
```

Result:

```
20Volvo
```

Try it Yourself »

JavaScript:

```
let x = "Volvo" + 16 + 4;
```

Result:

```
Volvo164
```

Try it Yourself »

In the first example, JavaScript treats 16 and 4 as numbers, until it reaches "Volvo".

In the second example, since the first operand is a string, all operands are treated as strings.

JavaScript Types are Dynamic

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

Example

```
let x;       // Now x is undefined
x = 5;       // Now x is a Number
x = "John";  // Now x is a String
```

Try it Yourself »

JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

Example

```
// Using double quotes:
let carName1 = "Volvo XC60";

// Using single quotes:
let carName2 = 'Volvo XC60';
```

Try it Yourself »

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example

```
// Single quote inside double quotes:
let answer1 = "It's alright";

// Single quotes inside double quotes:
let answer2 = "He is called 'Johnny'";
```

```
// Double quotes inside single quotes:
let answer3 = 'He is called "Johnny"';
```

Try it Yourself »

You will learn more about strings later in this tutorial.

JavaScript Numbers

All JavaScript numbers are stored as decimal numbers (floating point).

Numbers can be written with, or without decimals:

Example

```
// With decimals:
let x1 = 34.00;

// Without decimals:
let x2 = 34;
```

Try it Yourself »

Exponential Notation

Extra large or extra small numbers can be written with scientific (exponential) notation:

Example

```
let y = 123e5;    // 12300000
let z = 123e-5;   // 0.00123
```

Try it Yourself »

You will learn more about numbers later in this tutorial.

Note

Most programming languages have many number types:

Whole numbers (integers):
byte (8-bit), short (16-bit), int (32-bit), long (64-bit)

Real numbers (floating-piont):
float (32-bit), double (64-bit).

Javascript are always one type:
double (64-bit floating point).

JavaScript BigInt

All JavaScript numbers are stored in a a 64-bit floating-point format.

JavaScript BigInt is a new datatype (2020) that can be used to store integer values that are too big to be represented by a normal JavaScript Number.

Example

```
let x = BigInt("12345678901234567890123456789O");
```

Try it Yourself »

JavaScript Booleans

Booleans can only have two values: `true` or `false`.

Example

```
let x = 5;
let y = 5;
let z = 6;
(x == y)       // Returns true
(x == z)       // Returns false
```

Try it Yourself »

Booleans are often used in conditional testing.

You will learn more about conditional testing later in this tutorial.

JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called `cars`, containing three items (car names):

Example

```
const cars = ["Saab", "Volvo", "BMW"];
```

Try it Yourself »

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

You will learn more about **arrays** later in this tutorial.

JavaScript Objects

JavaScript objects are written with curly braces {}.

Object properties are written as name:value pairs, separated by commas.

Example

```
const person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
```

Try it Yourself »

The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

You will learn more about **objects** later in this tutorial.

The typeof Operator

You can use the JavaScript `typeof` operator to find the type of a JavaScript variable.

The typeof operator returns the type of a variable or an expression:

Example

```
typeof ""             // Returns "string"
typeof "John"         // Returns "string"
typeof "John Doe"     // Returns "string"
```

Try it Yourself »

Example

```
typeof 0              // Returns "number"
typeof 314            // Returns "number"
typeof 3.14           // Returns "number"
typeof (3)            // Returns "number"
typeof (3 + 4)        // Returns "number"
```

Try it Yourself »

You will learn more about **typeof** later in this tutorial.

Undefined

In JavaScript, a variable without a value, has the value undefined. The type is also undefined.

Example

```
let car;    // Value is undefined, type is undefined
```

Try it Yourself »

Any variable can be emptied, by setting the value to undefined. The type will also be undefined.

Example

```
car = undefined;    // Value is undefined, type is undefined
```

Try it Yourself »

Empty Values

An empty value has nothing to do with undefined.

An empty string has both a legal value and a type.

Example

```
let car = "";    // The value is "", the typeof is "string"
```

Try it Yourself »

Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- String Operators
- Logical Operators
- Bitwise Operators
- Ternary Operators
- Type Operators

JavaScript Arithmetic Operators

Arithmetic Operators are used to perform arithmetic on numbers:

Arithmetic Operators Example

```
let a = 3;
let x = (100 + 50) * a;
```

Try it Yourself »

Operator	Description
+	Addition
-	Subtraction

*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Arithmetic operators are fully described in the **JS Arithmetic** chapter.

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

The **Addition Assignment Operator** (+=) adds a value to a variable.

Assignment

```
let x = 10;
x += 5;
```

Try it Yourself »

Operator	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

Assignment operators are fully described in the **JS Assignment** chapter.

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	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

Shift Assignment Operators

Operator	Example	Same As
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y

Bitwise Assignment Operators

Operator	Example	Same As
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y

Logical Assignment Operators

Operator	Example	Same As
&&=	x &&= y	x = x && (x = y)
||=	x ||= y	x = x || (x = y)
??=	x ??= y	x = x ?? (x = y)

Note

The Logical assignment operators are ES2020.

The = Operator

The **Simple Assignment Operator** assigns a value to a variable.

Simple Assignment Examples

```
let x = 10;
```

Try it Yourself »

```
let x = 10 + y;
```

Try it Yourself »

The += Operator

The **Addition Assignment Operator** adds a value to a variable.

Addition Assignment Examples

```
let x = 10;
x += 5;
```

Try it Yourself »

```
let text = "Hello"; text += " World";
```

Try it Yourself »

The -= Operator

The **Subtraction Assignment Operator** subtracts a value from a variable.

Subtraction Assignment Example

```
let x = 10;
x -= 5;
```

Try it Yourself »

The *= Operator

The **Multiplication Assignment Operator** multiplies a variable.

Multiplication Assignment Example

```
let x = 10;
x *= 5;
```

Try it Yourself »

The **= Operator

The **Exponentiation Assignment Operator** raises a variable to the power of the operand.

Exponentiation Assignment Example

```
let x = 10;
x **= 5;
```

Try it Yourself »

The /= Operator

The **Division Assignment Operator** divides a variable.

Division Assignment Example

```
let x = 10;
x /= 5;
```

Try it Yourself »

The %= Operator

The **Remainder Assignment Operator** assigns a remainder to a variable.

Remainder Assignment Example

```
let x = 10;
x %= 5;
```

Try it Yourself »

The <<= Operator

The **Left Shift Assignment Operator** left shifts a variable.

Left Shift Assignment Example

```
let x = -100;
x <<= 5;
```

Try it Yourself »

The >>= Operator

The **Right Shift Assignment Operator** right shifts a variable (signed).

Right Shift Assignment Example

```
let x = -100;
x >>= 5;
```

Try it Yourself »

The >>>= Operator

The **Unsigned Right Shift Assignment Operator** right shifts a variable (unsigned).

Unsigned Right Shift Assignment Example

```
let x = -100;
x >>>= 5;
```

Try it Yourself »

The &= Operator

The **Bitwise AND Assignment Operator** does a bitwise AND operation on two operands and assigns the result to the the variable.

Bitwise AND Assignment Example

```
let x = 10;
x &= 5;
```

Try it Yourself »

The |= Operator

The **Bitwise OR Assignment Operator** does a bitwise OR operation on two operands and assigns the result to the variable.

Bitwise OR Assignment Example

```
let x = 10;
x |= 5;
```

Try it Yourself »

The ^= Operator

The **Bitwise XOR Assignment Operator** does a bitwise XOR operation on two operands and assigns the result to the variable.

Bitwise XOR Assignment Example

```
let x = 10;
x ^= 5;
```

Try it Yourself »

The &&= Operator

The **Logical AND assignment operator** is used between two values.

If the first value is true, the second value is assigned.

Logical AND Assignment Example

```
let x = 10;
x &&= 5;
```

Try it Yourself »

The `&&=` operator is an ES2020 feature.

The ||= Operator

The **Logical OR assignment operator** is used between two values.

If the first value is false, the second value is assigned.

Logical OR Assignment Example

```
let x = 10;
x ||= 5;
```

Try it Yourself »

The `||=` operator is an ES2020 feature.

The ??= Operator

The **Nullish coalescing assignment operator** is used between two values.

If the first value is undefined or null, the second value is assigned.

Nullish Coalescing Assignment Example

```
let x = 10;
x ??= 5;
```

Try it Yourself »

JavaScript Comparison Operators

Operator	Description
==	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

Comparison operators are fully described in the **JS Comparisons** chapter.

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that `x = 5`, the table below explains the comparison operators:

Operator	Description	Comparing	Returns	Try it
==	equal to	x == 8	false	Try it »
		x == 5	true	Try it »

		x == "5"	true	Try it »
===	equal value and equal type	x === 5	true	Try it »
		x === "5"	false	Try it »
!=	not equal	x != 8	true	Try it »
!==	not equal value or not equal type	x !== 5	false	Try it »
		x !== "5"	true	Try it »
		x !== 8	true	Try it »
>	greater than	x > 8	false	Try it »
<	less than	x < 8	true	Try it »
>=	greater than or equal to	x >= 8	false	Try it »
<=	less than or equal to	x <= 8	true	Try it »

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that x = 6 and y = 3, the table below explains the logical operators:

Operator	Description	Example	Try it
&&	and	(x < 10 && y > 1) is true	Try it »
||	or	(x == 5 || y == 5) is false	Try it »
!	not	!(x == y) is true	Try it »

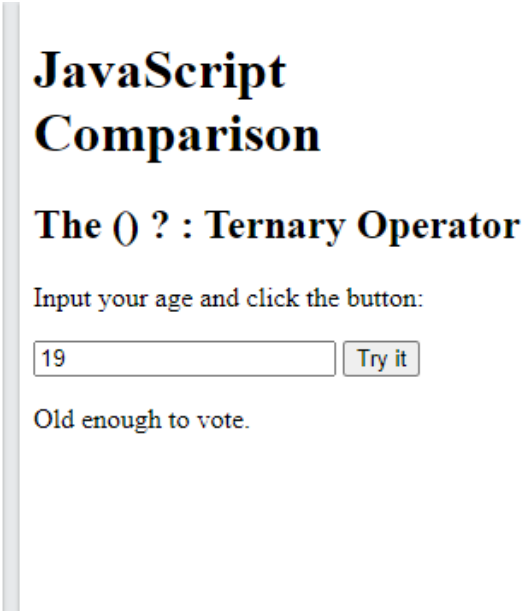
Conditional (Ternary) Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

variablename = (*condition*) ? *value1*:*value2*

```
<!DOCTYPE html>
<html><body>
<h1>JavaScript Comparison</h1>
<h2>The () ? : Ternary Operator</h2>
<p>Input your age and click the button:</p>
<input id="age" value="18" />
<button onclick="myFunction()">Try
it</button>
<p id="demo"></p>
<script>
function myFunction() {
  let age =
document.getElementById("age").value;
  let voteable = (age < 18) ? "Too
young":"Old enough";
document.getElementById("demo").innerHTML =
voteable + " to vote.";
}
</script></body></html>
```



Adding JavaScript Strings

The + operator can also be used to add (concatenate) strings.

Example

```
let text1 = "John";
let text2 = "Doe";
let text3 = text1 + " " + text2;
```

The result of text3 will be:

```
John Doe
```

Try it Yourself »

The += assignment operator can also be used to add (concatenate) strings:

Example

```
let text1 = "What a very ";
text1 += "nice day";
```

The result of text1 will be:

```
What a very nice day
```

Try it Yourself »

When used on strings, the + operator is called the concatenation operator.

Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

Example

```
let x = 5 + 5;
let y = "5" + 5;
let z = "Hello" + 5;
```

The result of *x*, *y*, and *z* will be:

```
10
55
Hello5
```

Try it Yourself »

If you add a number and a string, the result will be a string!

JavaScript Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

Logical operators are fully described in the **JS Comparisons** chapter.

JavaScript Type Operators

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

Type operators are fully described in the **JS Type Conversion** chapter.

- Converting Strings to Numbers
- Converting Numbers to Strings
- Converting Dates to Numbers
- Converting Numbers to Dates
- Converting Booleans to Numbers
- Converting Numbers to Booleans

JavaScript Bitwise Operators

Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

Operator	Description	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

The examples above uses 4 bits unsigned examples. But JavaScript uses 32-bit signed numbers.
Because of this, in JavaScript, ~ 5 will not return 10. It will return -6.
~00000000000000000000000000000101 will return
11111111111111111111111111111010

Bitwise operators are fully described in the **JS Bitwise** chapter.

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- Use `if` to specify a block of code to be executed, if a specified condition is true
- Use `else` to specify a block of code to be executed, if the same condition is false
- Use `else if` to specify a new condition to test, if the first condition is false
- Use `switch` to specify many alternative blocks of code to be executed

The `switch` statement is described in the next chapter.

The if Statement

Use the `if` statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

```
if (condition) {
  //  block of code to be executed if the condition is true
}
```

Note that `if` is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error.

Example

Make a "Good day" greeting if the hour is less than 18:00:

```
if (hour < 18) {
  greeting = "Good day";
}
```

The result of greeting will be:

```
Good day
```

Try it Yourself »

The else Statement

Use the `else` statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

Example

If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

```
if (hour < 18) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

The result of greeting will be:

```
Good day
```

Try it Yourself »

The else if Statement

Use the `else if` statement to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and
condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and
```

```
condition2 is false
}
```

Example

If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

The result of greeting will be:

```
Good day
```

Try it Yourself »

JavaScript Switch Statement

❮ PreviousNext ❯

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.

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
```

```
  default:
    // code block
}
```

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.

Example

The `getDay()` method returns the weekday as a number between 0 and 6.

(Sunday=0, Monday=1, Tuesday=2 ..)

This example uses the weekday number to calculate the weekday name:

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
}
```

The result of day will be:

```
Wednesday
```

Try it Yourself »

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

You can write:

```
for (let i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```

Try it Yourself »

Different Kinds of Loops

JavaScript supports different kinds of loops:

- `for` - loops through a block of code a number of times
- `for/in` - loops through the properties of an object
- `for/of` - loops through the values of an iterable object
- `while` - loops through a block of code while a specified condition is true
- `do/while` - also loops through a block of code while a specified condition is true

The For Loop

The `for` statement creates a loop with 3 optional expressions:

```
for (expression 1; expression 2; expression 3) {
  // code block to be executed
}
```

Expression 1 is executed (one time) before the execution of the code block.

Expression 2 defines the condition for executing the code block.

Expression 3 is executed (every time) after the code block has been executed.

Example

```
for (let i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
```

Try it Yourself »

From the example above, you can read:

Expression 1 sets a variable before the loop starts (let i = 0).

Expression 2 defines the condition for the loop to run (i must be less than 5).

Expression 3 increases a value (i++) each time the code block in the loop has been executed.

Expression 1

Normally you will use expression 1 to initialize the variable used in the loop (let i = 0).

This is not always the case. JavaScript doesn't care. Expression 1 is optional.

You can initiate many values in expression 1 (separated by comma):

Example

```
for (let i = 0, len = cars.length, text = ""; i < len; i++) {
  text += cars[i] + "<br>";
}
```

Try it Yourself »

And you can omit expression 1 (like when your values are set before the loop starts):

Example

```
let i = 2;
let len = cars.length;
```

```
let text = "";
for (; i < len; i++) {
  text += cars[i] + "<br>";
}
```

Try it Yourself »

Expression 2

Often expression 2 is used to evaluate the condition of the initial variable.

This is not always the case. JavaScript doesn't care. Expression 2 is also optional.

If expression 2 returns true, the loop will start over again. If it returns false, the loop will end.

If you omit expression 2, you must provide a **break** inside the loop. Otherwise the loop will never end. This will crash your browser. Read about breaks in a later chapter of this tutorial.

Expression 3

Often expression 3 increments the value of the initial variable.

This is not always the case. JavaScript doesn't care. Expression 3 is optional.

Expression 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else.

Expression 3 can also be omitted (like when you increment your values inside the loop):

Example

```
let i = 0;
let len = cars.length;
let text = "";
for (; i < len; ) {
  text += cars[i] + "<br>";
  i++;
}
```

Try it Yourself »

Loop Scope

Using `var` in a loop:

Example

```
var i = 5;

for (var i = 0; i < 10; i++) {
  // some code
}

// Here i is 10
```

Try it Yourself »

Using `let` in a loop:

Example

```
let i = 5;

for (let i = 0; i < 10; i++) {
  // some code
}

// Here i is 5
```

Try it Yourself »

In the first example, using `var`, the variable declared in the loop redeclares the variable outside the loop.

In the second example, using `let`, the variable declared in the loop does not redeclare the variable outside the loop.

When `let` is used to declare the i variable in a loop, the i variable will only be visible within the loop.

The For In Loop

The JavaScript `for in` statement loops through the properties of an Object:

Syntax

```
for (key in object) {
  // code block to be executed
}
```

Example

```
const person = {fname:"John", lname:"Doe", age:25};

let text = "";
for (let x in person) {
  text += person[x];
}
```

Try it Yourself »

Example Explained

- The **for in** loop iterates over a **person** object
- Each iteration returns a **key** (x)
- The key is used to access the **value** of the key
- The value of the key is **person[x]**

For In Over Arrays

The JavaScript `for in` statement can also loop over the properties of an Array:

Syntax

```
for (variable in array) {
  code
}
```

Example

```
const numbers = [45, 4, 9, 16, 25];

let txt = "";
for (let x in numbers) {
  txt += numbers[x];
}
```

Try it Yourself »

The For Of Loop

The JavaScript `for of` statement loops through the values of an iterable object.

It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more:

Syntax

```
for (variable of iterable) {
  // code block to be executed
}
```

variable - For every iteration the value of the next property is assigned to the variable. *Variable* can be declared with `const`, `let`, or `var`.

iterable - An object that has iterable properties.

Looping over an Array

Example

```
const cars = ["BMW", "Volvo", "Mini"];

let text = "";
for (let x of cars) {
  text += x;
}
```

Try it Yourself »

Looping over a String

Example

```
let language = "JavaScript";

let text = "";
for (let x of language) {
text += x;
}
```

Try it Yourself »

The While Loop

The `while` loop loops through a block of code as long as a specified condition is true.

Syntax

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

Example

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

Example

```
while (i < 10) {
  text += "The number is " + i;
  i++;
}
```

Try it Yourself »

If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser.

The Do While Loop

The `do while` loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

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

Example

The example below uses a `do while` loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Example

```
do {
  text += "The number is " + i;
  i++;
}
while (i < 10);
```

Try it Yourself »

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

```
window.alert("sometext");
```

The `window.alert()` method can be written without the window prefix.

Example

```
alert("I am an alert box!");
```

Try it Yourself »

Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns **true**. If the user clicks "Cancel", the box returns **false**.

Syntax

```
window.confirm("sometext");
```

The `window.confirm()` method can be written without the window prefix.

Example

```
if (confirm("Press a button!")) {
  txt = "You pressed OK!";
} else {
  txt = "You pressed Cancel!";
}
```

Try it Yourself »

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

```
window.prompt("sometext","defaultText");
```

The `window.prompt()` method can be written without the window prefix.

Example

```
let person = prompt("Please enter your name", "Harry Potter");
let text;
if (person == null || person == "") {
  text = "User cancelled the prompt.";
} else {
  text = "Hello " + person + "! How are you today?";
}
```

Try it Yourself »

Line Breaks

To display line breaks inside a popup box, use a back-slash followed by the character n.

Example

```
alert("Hello\nHow are you?");
```

Try it Yourself »

JavaScript Events

❮ PreviousNext ❯

HTML events are **"things"** that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

```
<element event='some JavaScript'>
```

With double quotes:

```
<element event="some JavaScript">
```

In the following example, an `onclick` attribute (with code), is added to a `<button>` element:

Example

```
<button onclick="document.getElementById('demo').innerHTML =
Date()">The time is?</button>
```

Try it Yourself »

In the example above, the JavaScript code changes the content of the element with id="demo".

In the next example, the code changes the content of its own element (using **`this`**`.innerHTML`):

Example

```
<button onclick="this.innerHTML = Date()">The time is?</button>
```

Try it Yourself »

JavaScript code is often several lines long. It is more common to see event attributes calling functions:

Example

```
<button onclick="displayDate()">The time is?</button>
```

Try it Yourself »

Common HTML Events

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

JavaScript Event Handlers

Event handlers can be used to handle and verify user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

Many different methods can be used to let JavaScript work with events:

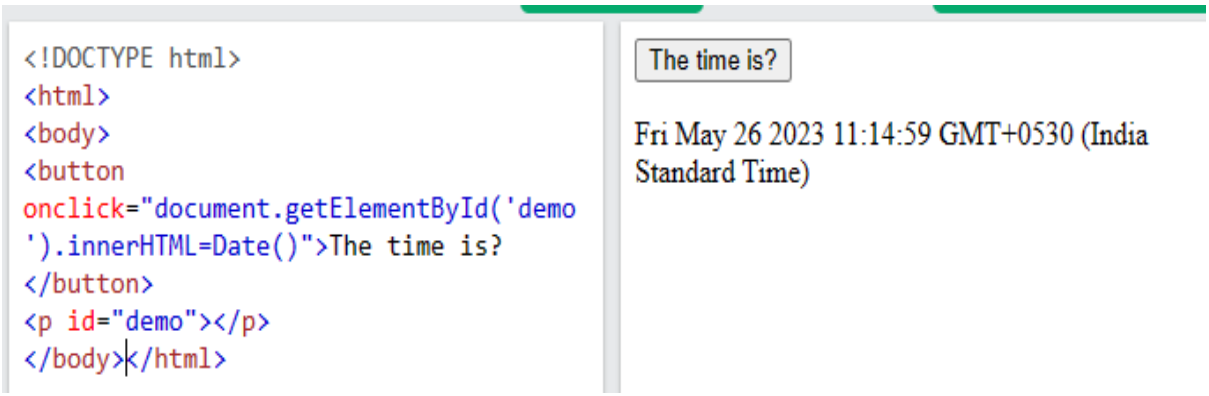
- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions
- You can assign your own event handler functions to HTML elements
- You can prevent events from being sent or being handled
- And more ...

These are some javascript events:

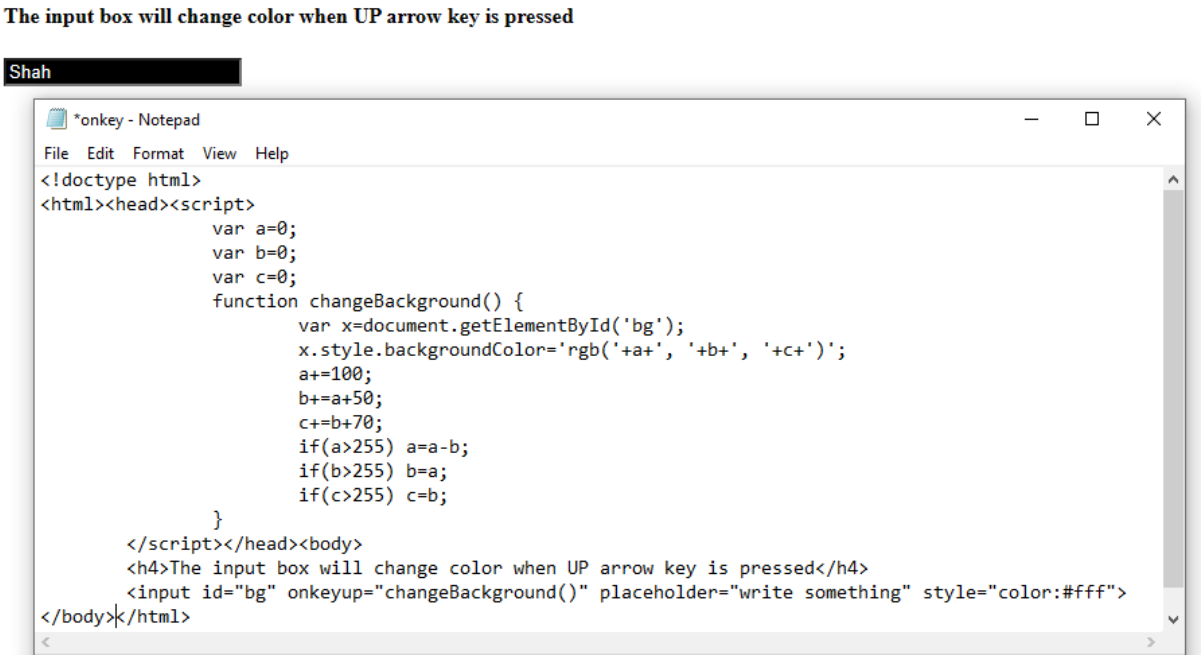
JavaScript onclick events: This is a mouse event and provokes any logic defined if the user clicks on the element it is bound to.

Example: In this example, we will display a message in the alert box when the button is clicked

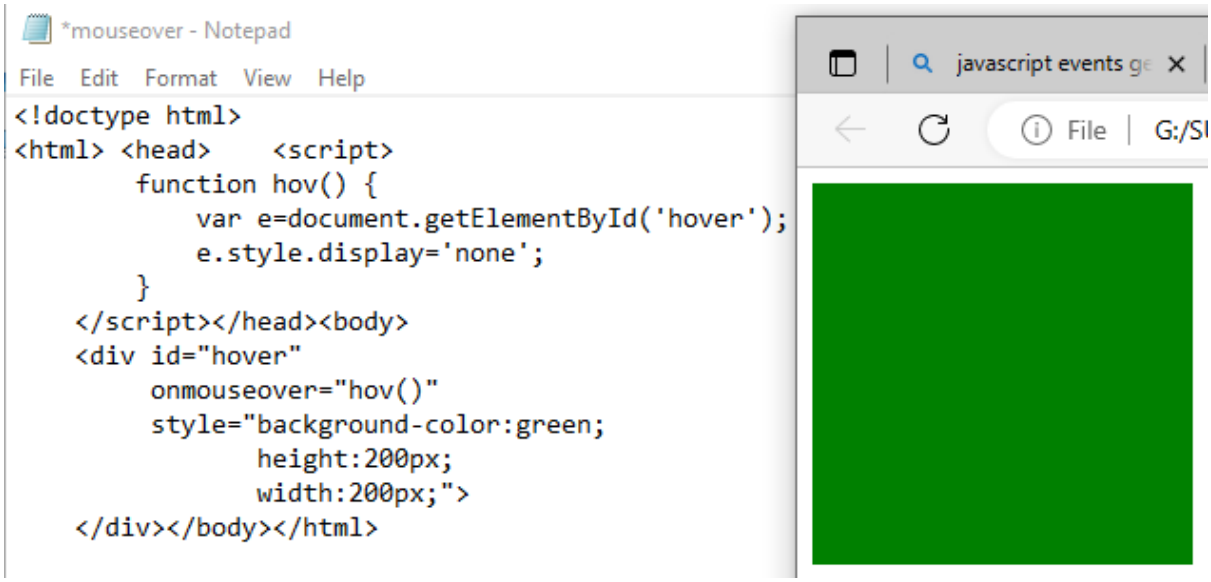
```
<!DOCTYPE html>
<html>
<body>
<button
onclick="document.getElementById('demo
').innerHTML=Date()">The time is?
</button>
<p id="demo"></p>
</body></html>
```



JavaScript onkeyup event: This event is a keyboard event and executes instructions whenever a key is released after pressing.
Example: In this example, we will change the color by pressing UP arrow key



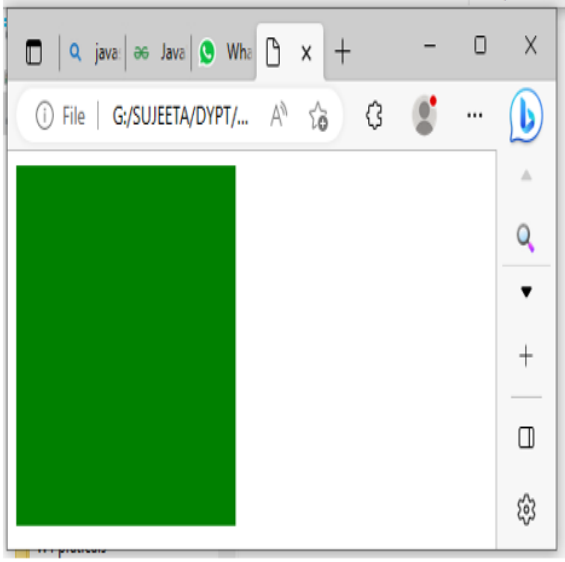
onmouseover event: This event corresponds to hovering the mouse pointer over the element and its children, to which it is bound to.
Example: In this example, we will make the box vanish when the mouse will be hovered on it



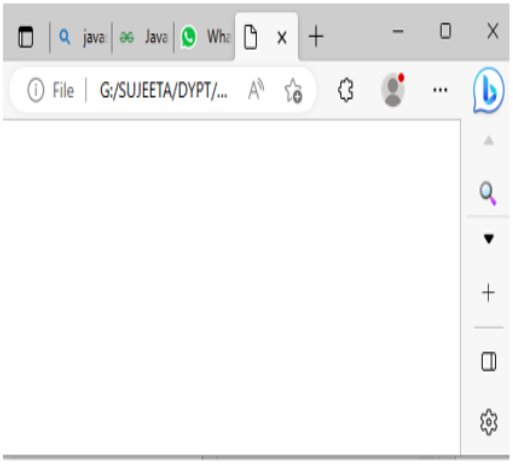
JavaScript onmouseout event: Whenever the mouse cursor leaves the element which handles a mouseout event, a function associated with it is executed.

```
File  Edit  Format  View  Help
<!doctype html>
<html><head><script>
        function out() {
            var e=document.getElementById('hover');
            e.style.display='none';
        }
    </script></head><body>
    <div id="hover" onmouseout="out()" style="background-color:green;height:200px;width:200px;">
    </div></body></html>
```

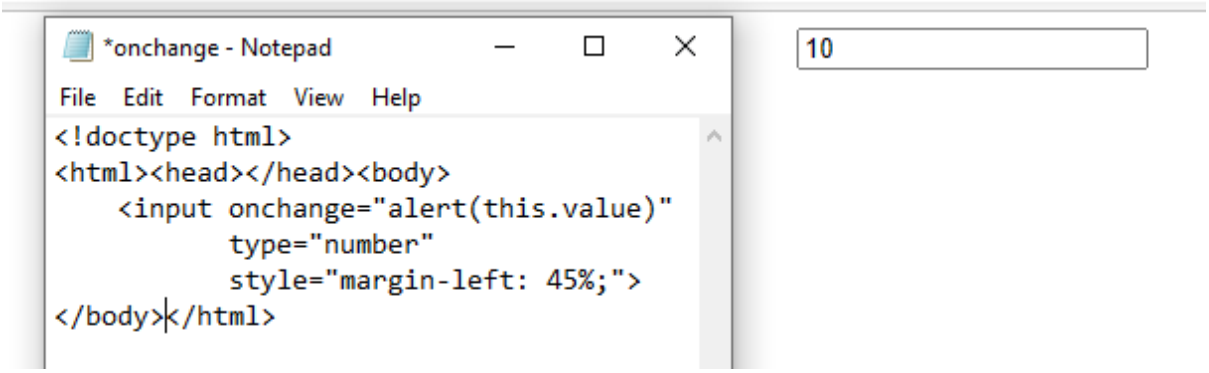
Ln 9, Col 18 100% Windows (CRLF) UTF-8



After Mouseout



JavaScript onchange event: This event detects the change in value of any element listing to this event.



JavaScript onload event: When an element is loaded completely, this event is evoked.

*onload - Notepad

File Edit Format View Help

```
<!doctype html>
<html><head></head><body>
    <img onload="alert('Image completely loaded')"
         alt="GFG-Logo"
         src="C:\Users\DELL\Desktop\download.jpg">
</body></html>
```

Ln 6, Col 8 100% Windows (CR

File | G:/SUJEETA/DYPT/MCA-I-SEM-II-AY-2023-23/Web%20Techno...

This page says

Image completely loaded

OK

JavaScript onfocus event: An element listing to this event executes instructions whenever it receives focus.

onfocus - Notepad

File Edit Format View Help

```
<!doctype html>
<!doctype html>
<html> <head>    <script>
        function focused() {
            var e=document.getElementById('inp');
            if(confirm('Got it?')) {
                e.blur();
            }
        }
    </script></head> <body>
    <p style="margin-left: 45%;">
        Take the focus into the input box below:
    </p>
    <input id="inp"
           onfocus="focused()""
           style=" margin-left: 45%;">
</body></html>
```

Ln 17, Col 8 100% Windows (CRLF

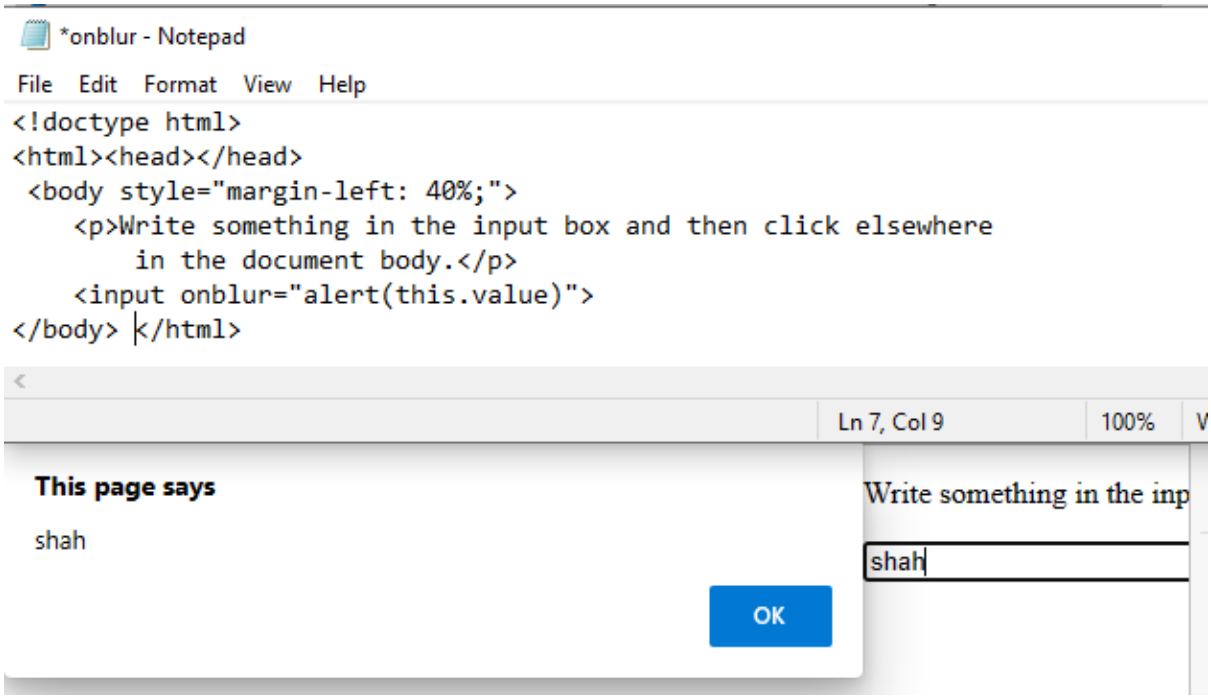
This page says

Got it?

OK Cancel

Take the focus into the input box belo

JavaScript onblur event: This event is evoked when an element loses focus.



PS: JavaScript onmouseup event listens to left and middle mouse clicks, but **onmousedown** event listens to left, middle, and right mouse clicks whereas **onclick** only handles left clicks.

JavaScript Arrays

❮ PreviousNext ❯

An array is a special variable, which can hold more than one value:

```
const cars = ["Saab", "Volvo", "BMW"];
```

Try it Yourself »

Why Use Arrays?

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
let car1 = "Saab";
let car2 = "Volvo";
let car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

const *array_name* = [*item1*, *item2*, ...];

It is a common practice to declare arrays with the const keyword.

Learn more about const with arrays in the chapter: JS Array Const.

Example

```
const cars = ["Saab", "Volvo", "BMW"];
```

Try it Yourself »

Spaces and line breaks are not important. A declaration can span multiple lines:

Example

```
const cars = [
  "Saab",
  "Volvo",
  "BMW"
];
```

Try it Yourself »

You can also create an array, and then provide the elements:

Example

```
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

Try it Yourself »

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

Example

```
const cars = new Array("Saab", "Volvo", "BMW");
```

Try it Yourself »

The two examples above do exactly the same.

There is no need to use `new Array()`.

For simplicity, readability and execution speed, use the array literal method.

Accessing Array Elements

You access an array element by referring to the **index number**:

```
const cars = ["Saab", "Volvo", "BMW"];
let car = cars[0];
```

Try it Yourself »

Note: Array indexes start with 0.

[0] is the first element. [1] is the second element.

Changing an Array Element

This statement changes the value of the first element in `cars`:

```
cars[0] = "Opel";
```

Example

```
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
```

Try it Yourself »

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

Example

```
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

Try it Yourself »

Arrays are Objects

Arrays are a special type of objects. The `typeof` operator in JavaScript returns "object" for arrays.

But, JavaScript arrays are best described as arrays.

Arrays use **numbers** to access its "elements". In this example, `person[0]` returns John:

Array:

```
const person = ["John", "Doe", 46];
```

Try it Yourself »

Objects use **names** to access its "members". In this example, `person.firstName` returns John:

Object:

```
const person = {firstName:"John", lastName:"Doe", age:46};
```

Try it Yourself »

Converting Arrays to Strings

The JavaScript method `toString()` converts an array to a string of (comma separated) array values.

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Result:

Banana,Orange,Apple,Mango

Try it Yourself »

The `join()` method also joins all array elements into a string.

It behaves just like `toString()`, but in addition you can specify the separator:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Result:

Banana * Orange * Apple * Mango

Try it Yourself »

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array.

JavaScript Array pop()

The `pop()` method removes the last element from an array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
```

Try it Yourself »

The `pop()` method returns the value that was "popped out":

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.pop();
```

Try it Yourself »

JavaScript Array push()

The `push()` method adds a new element to an array (at the end):

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");
```

Try it Yourself »

The `push()` method returns the new array length:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.push("Kiwi");
```

Try it Yourself »

Shifting Elements

Shifting is equivalent to popping, but working on the first element instead of the last.

JavaScript Array shift()

The `shift()` method removes the first array element and "shifts" all other elements to a lower index.

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();
```

Try it Yourself »

The `shift()` method returns the value that was "shifted out":

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.shift();
```

Try it Yourself »

JavaScript Array unshift()

The `unshift()` method adds a new element to an array (at the beginning), and "unshifts" older elements:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```

Try it Yourself »

The `unshift()` method returns the new array length:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```

Try it Yourself »

Changing Elements

Array elements are accessed using their **index number**:

Array **indexes** start with 0:

[0] is the first array element
[1] is the second
[2] is the third ...

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";
```

Try it Yourself »

JavaScript Array delete()

Warning !

Array elements can be deleted using the JavaScript operator `delete`.

Using `delete` leaves `undefined` holes in the array.

Use pop() or shift() instead.

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];
```

Try it Yourself »

Merging (Concatenating) Arrays

The `concat()` method creates a new array by merging (concatenating) existing arrays:

Example (Merging Two Arrays)

```
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];

const myChildren = myGirls.concat(myBoys);
```

Try it Yourself »

The `concat()` method does not change the existing arrays. It always returns a new array.

The `concat()` method can take any number of array arguments:

Example (Merging Three Arrays)

```
const arr1 = ["Cecilie", "Lone"];
const arr2 = ["Emil", "Tobias", "Linus"];
const arr3 = ["Robin", "Morgan"];
const myChildren = arr1.concat(arr2, arr3);
```

Try it Yourself »

The `concat()` method can also take strings as arguments:

Example (Merging an Array with Values)

```
const arr1 = ["Emil", "Tobias", "Linus"];
const myChildren = arr1.concat("Peter");
```

Try it Yourself »

Splicing and Slicing Arrays

The `splice()` method adds new items to an array.

The `slice()` method slices out a piece of an array.

JavaScript Array splice()

The `splice()` method can be used to add new items to an array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

Try it Yourself »

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

The `splice()` method returns an array with the deleted items:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 2, "Lemon", "Kiwi");
```

Try it Yourself »

Using splice() to Remove Elements

With clever parameter setting, you can use `splice()` to remove elements without leaving "holes" in the array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1);
```

Try it Yourself »

The first parameter (0) defines the position where new elements should be **added** (spliced in).

The second parameter (1) defines **how many** elements should be **removed**.

The rest of the parameters are omitted. No new elements will be added.

JavaScript Array slice()

The `slice()` method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

Example

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1);
```

Try it Yourself »

Note

The `slice()` method creates a new array.

The `slice()` method does not remove any elements from the source array.

This example slices out a part of an array starting from array element 3 ("Apple"):

Example

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(3);
```

Try it Yourself »

The `slice()` method can take two arguments like `slice(1, 3)`.

The method then selects elements from the start argument, and up to (but not including) the end argument.

Example

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1, 3);
```

Try it Yourself »

If the end argument is omitted, like in the first examples, the `slice()` method slices out the rest of the array.

Example

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(2);
```

Try it Yourself »

JavaScript Objects

❮ PreviousNext ❯

Real Life Objects, Properties, and Methods

In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

Object	Properties
	car.name = Fiat car.model = 500 car.weight = 850kg car.color = white

All cars have the same **properties**, but the property **values** differ from car to car.

All cars have the same **methods**, but the methods are performed **at different times**.

JavaScript Objects

You have already learned that JavaScript variables are containers for data values.

This code assigns a **simple value** (Fiat) to a **variable** named car:

```
let car = "Fiat";
```

Try it Yourself »

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

```
const car = {type:"Fiat", model:"500", color:"white"};
```

Try it Yourself »

The values are written as **name:value** pairs (name and value separated by a colon).

It is a common practice to declare objects with the `const` keyword.

Learn more about using `const` with objects in the chapter: JS Const.

Object Definition

You define (and create) a JavaScript object with an object literal:

Example

```
const person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
```

Try it Yourself »

Spaces and line breaks are not important. An object definition can span multiple lines:

Example

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

Try it Yourself »

Object Properties

The **name:values** pairs in JavaScript objects are called **properties**:

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

Accessing Object Properties

You can access object properties in two ways:

objectName.propertyName

or

objectName["propertyName"]

Example1

```
person.lastName;
```

Try it Yourself »

Example2

```
person["lastName"];
```

Try it Yourself »

JavaScript objects are containers for **named values** called properties.

Object Methods

Objects can also have **methods**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

A method is a function stored as a property.

Example

```
const person = {
  firstName: "John",
```

```
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

In the example above, `this` refers to the **person object**.

I.E. **this.firstName** means the **firstName** property of **this**.

I.E. **this.firstName** means the **firstName** property of **person**.

What is this?

In JavaScript, the `this` keyword refers to an **object**.

Which object depends on how `this` is being invoked (used or called).

The `this` keyword refers to different objects depending on how it is used:

In an object method, `this` refers to the **object**.
Alone, `this` refers to the **global object**.
In a function, `this` refers to the **global object**.
In a function, in strict mode, `this` is `undefined`.
In an event, `this` refers to the **element** that received the event.

Methods like `call()`, `apply()`, and `bind()` can refer `this` to **any object**.

Note

`this` is not a variable. It is a keyword. You cannot change the value of `this`.

See Also:

The JavaScript **this** Tutorial

The this Keyword

In a function definition, `this` refers to the "owner" of the function.

In the example above, `this` is the **person object** that "owns" the `fullName` function.

In other words, `this.firstName` means the `firstName` property of **this object**.

Learn more about `this` in The JavaScript this Tutorial.

Accessing Object Methods

You access an object method with the following syntax:

objectName.methodName()

Example

```
name = person.fullName();
```

Try it Yourself »

If you access a method **without** the () parentheses, it will return the **function definition**:

Example

```
name = person.fullName;
```

Try it Yourself »

Do Not Declare Strings, Numbers, and Booleans as Objects!

When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

```
x = new String();        // Declares x as a String object
y = new Number();        // Declares y as a Number object
z = new Boolean();       // Declares z as a Boolean object
```

Avoid `String`, `Number`, and `Boolean` objects. They complicate your code and slow down execution speed.

JavaScript Functions

❮ PreviousNext ❯

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

Example

```
// Function to compute the product of p1 and p2
function myFunction(p1, p2) {
  return p1 * p2;
}
```

Try it Yourself »

JavaScript Function Syntax

A JavaScript function is defined with the `function` keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas: **(*parameter1, parameter2, ...*)**

The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

You will learn a lot more about function invocation later in this tutorial.

Function Return

When JavaScript reaches a `return` statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

Example

Calculate the product of two numbers, and return the result:

```
// Function is called, the return value will end up in x
let x = myFunction(4, 3);

function myFunction(a, b) {
// Function returns the product of a and b
  return a * b;
}
```

Try it Yourself »

Why Functions?

With functions you can reuse code

You can write code that can be used many times.

You can use the same code with different arguments, to produce different results.

The () Operator

The () operator invokes (calls) the function:

Example

Convert Fahrenheit to Celsius:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius(77);
```

Try it Yourself »

Accessing a function with incorrect parameters can return an incorrect answer:

Example

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius();
```

Try it Yourself »

Accessing a function without () returns the function and not the function result:

Example

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius;
```

Try it Yourself »

Note

As you see from the examples above, `toCelsius` refers to the function object, and `toCelsius()` refers to the function result.

Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

Example

Instead of using a variable to store the return value of a function:

```
let x = toCelsius(77);
let text = "The temperature is " + x + " Celsius";
```

You can use the function directly, as a variable value:

```
let text = "The temperature is " + toCelsius(77) + " Celsius";
```

Try it Yourself »

You will learn a lot more about functions later in this tutorial.

Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.

Local variables can only be accessed from within the function.

Example

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Try it Yourself »

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

6 Examples of JavaScript Applications

JavaScript, when used at either the front-end or back-end or even for full-stack development, offers developers a diverse range of applications.

JavaSript in realtime- 6 Examples of JavaScript: Where and When to Use - Trio | Trio Developers

The following JavaScript examples explain in detail the varying uses of JavaScript.

1. Presentations

Developers have the option of recruiting two JavaScript libraries, RevealJS and BespokeJS, to generate a slide deck on the web.

RevealJS is an HTML presentation framework that implements touch into its programming.

Hence, online presentations can be accessed by those with mobile devices like phones and tablets. This framework supports all CSS color formats as well as miscellaneous themes, transition styles, and backgrounds.

The BespokeJS plugin is a feature-heavy framework that supplies rich properties such as scaling, animated bullets, and syntax highlighting whilst coding. BespokeJS is characterized as being lightweight.

2. Web Development

Web development involves all the behaviors enlisted to create a dynamic and interactive web page.

In addition to performing web page interactions, JavaScript can open PDFs, run widgets, and load web page content in the absence of a refresh.

3. Server Applications

Node.js is the most frequently used runtime environment for JavaScript, where developers can write, test, and debug code. Through Node.js you can write server-side software.

An example of a JavaScript server application is the Opera Unite feature of the Opera Browser.

Opera Unite lets users run server applications like file sharing and streaming straight from the web browser.

4. Web Applications

Angular and Vue.js are popular JavaScript frameworks that developers make use of during app development.

Netflix and PayPal were developed with AngularJS and APIs.

An application programming interface (API) is a protocol for accessing web-based software.

5. Games

Games in JavaScript tend to harness the EaselJS library, a library known for its rich graphics. JavaScript and HTML5 are a favored combo for creating games on the web.

HTML5 is designed so that you have full access to the web unaccompanied by additional plugins like Flash.

To that end, it's cross-platform, so you won't have to bother with switching devices to get a full web page.

6. Mobile Applications

Mobile applications are built to be stand-alone apps void of any web-based context. JavaScript developers look to React Native and ReactJS for this type of development.

- **React Native**

React Native serves the purpose of mobile app building well as it specifically supports the implementation of native features into hybrid apps. Though hybrid apps use web technologies, they can be launched from a mobile app platform without opening a web browser.

- **Virtual DOM**

ReactJS is responsible for building user interfaces. It is often used for the base of mobile apps.

The document object model (DOM) is a chief component of ReactJS. Consider DOM as an API for HTML and XML documents, allowing the user to read and manipulate the content on a page.

Virtual DOM (VDOM) entrusts the cache to hold a virtual instance of the DOM in memory. VDOM works in place of DOM as a less intensive clone where developers can edit a single component of the DOM at a time.

Only the changes will be synced with the original DOM. In refusing to render the real DOM, VDOM solves the issue of building large applications where data changes often, speeding up operations as a whole.

- **Single-way data flow**

Single-way data flow, alternatively called unidirectional data flow, is a feature of React.

This type of data flow refers to when data is transferred from one part of an application to the others in only one way.

Developers have better control of the flow of data with single-way data flow versus two-way data flow in which the values passed to a component are mutable.

https://www.w3resource.com/javascript/form/javascript-sample-registration-form-validation.php