

UNIT-IV

Introduction to Client Side Scripting , Introduction to Java Script , JavaScript Types , Variables in JS, Operators in JS , Conditions Statements , Java Script Loops, JS Popup Boxes , JS Events , JS Arrays, Working with Arrays, JS Objects ,JS Functions , Using Java Script in Real time ,Validation of Forms, Related Examples

Introduction to Client Side Scripting:

1. Web Programming

The development of conventional software requires knowledge of traditional programming language in which the software is to be written. The well-known programming languages include C/C++ and Java which are high level computer languages. The syntax of these languages is similar to English like statements which require expertise of computer programmer's specific to syntax and semantics of a particular language. Using the syntax, a computer programmer writes a collection of instructions that performs a specific task. These instructions are called computer program. A computer program written in any high-level language is not suitable for web development because of the diverse nature of web applications.

Web development is associated with a different kind of programming paradigm called web programming. A person who programs a website is called a web programmer. It is the area which is concerned with writing of source code (computer program) to create a website. Web programming is sometimes called scripting. The source code is written in some scripting language. Web applications require multi-platform accessibility and employ a hypermedia paradigm. It may be static or dynamic depending upon the requirement of an organization.

1.1 Client Side Scripting

The client side environment enables interaction within the web browser. The source code is downloaded from web server to client's computer and executed by the browser. An example of a client side execution is a mouse over effect (typically triggered while choosing a navigation option), which runs on the user's machine and not on the server. The language JavaScript is used to script client side codes for web applications.

1.2 Server Side Scripting

The server side environment enables code to execute on the server machine. The requests of users are sent to server which executes a piece of code, generates output and sends the response back to the client. This response is utilized by the client computer to display the result for the user.

1.3 Static and Dynamic Websites

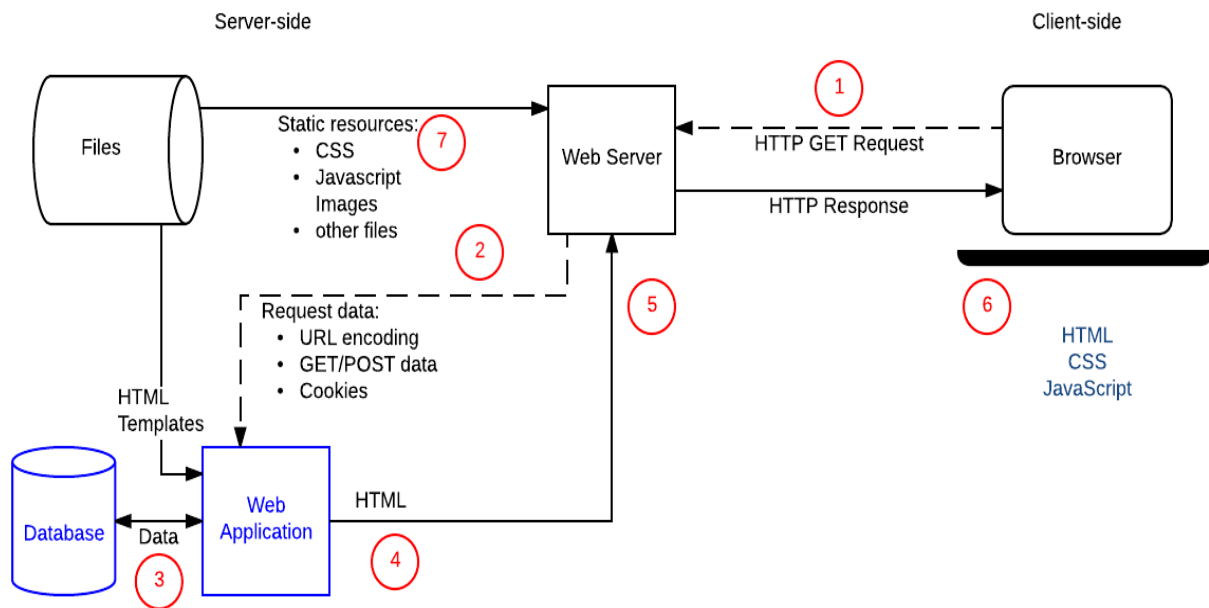
A static website is the simplest one website that does not allow frequent change in the contents. It is written in a plain HTML which displays the text on user computer using the web browser.

A dynamic website is more complex where contents of website are changed frequently. It is written in HTML and some scripting language which dynamically updates the contents being requested by the user.

Introduction to Client-side Scripting:

Scripts in Service Now fall into two categories:

- **Client-side**
- **Server-side**



This module is about client-side scripting. Client-side scripts execute within a user's browser and are used to manage forms and form fields. Examples of things client-side scripts can do include:

- Place the cursor in a form field on form load
- Generate alerts, confirmations, and messages
- Populate a form field in response to another field's value
- Highlight a form field
- Validate form data
- Modify choice list options
- Hide/Show fields or sections

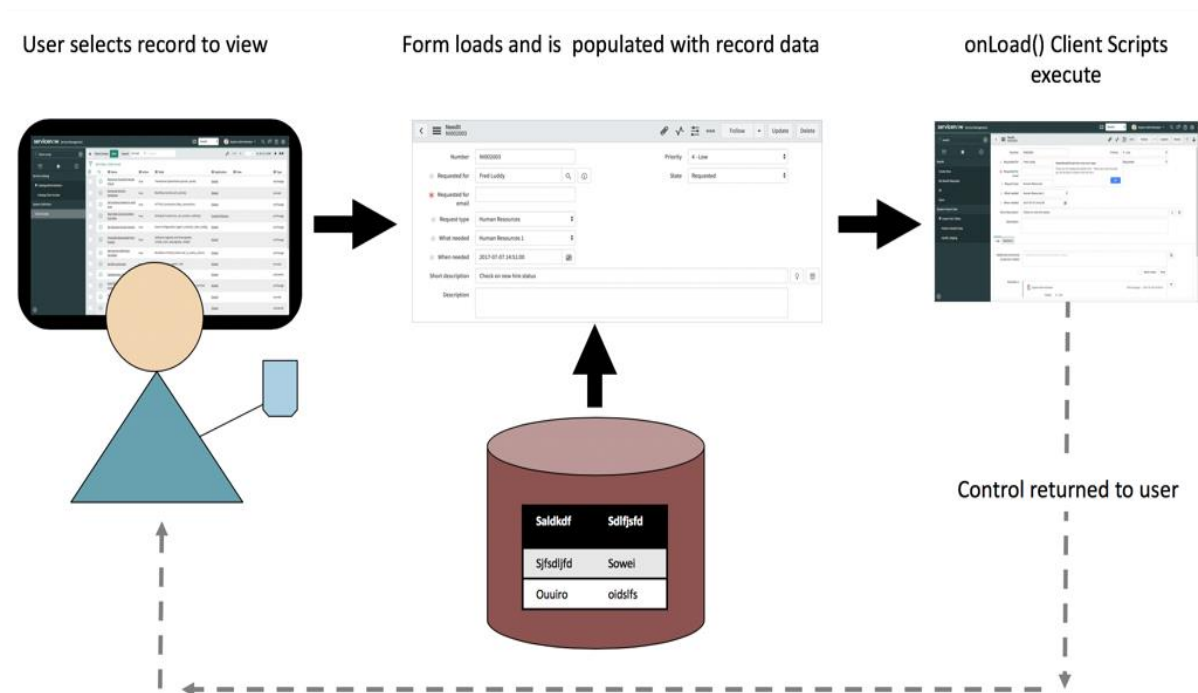
Client Script Types

A Client Script executes client-side script logic when forms are:

- **Loaded**
- **Changed**
- **Submitted**

a) onLoad

onLoad Client Scripts execute script logic when forms are loaded. Uses *onload* Client Scripts to manipulate a form's appearance or content. For example, setting field or form-level messages based on the presence of a value. Uses *onLoad* Client Scripts sparingly as they impact form load times.



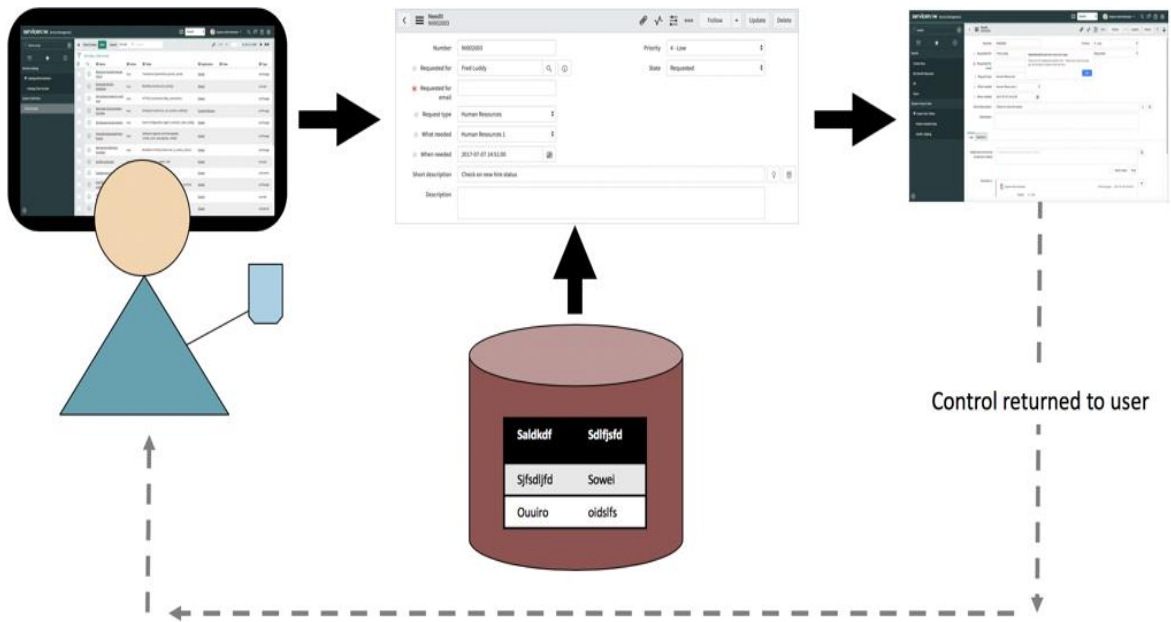
b) onChange:

onChange Client Scripts execute script logic when a particular field's value changes. Use *onChange* Client Scripts to respond to field values of interest and to modify another field's value or attributes. For example, if the *State* field's value changes to *Closed Complete*, generate an alert and make the *Description* field mandatory.

User selects record to view

Form loads and is populated with record data

onLoad() Client Scripts execute



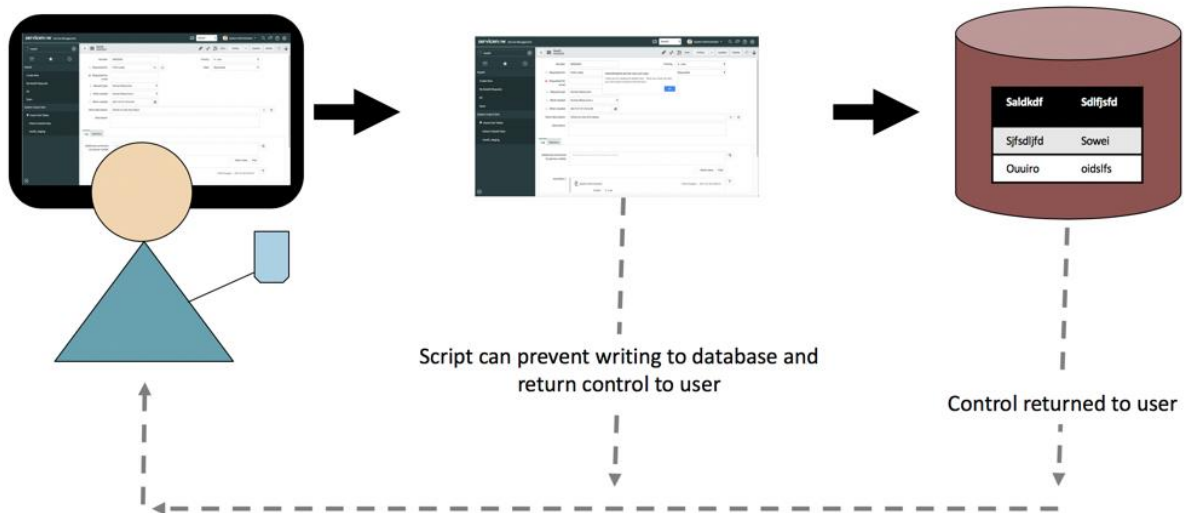
c) onSubmit:

onSubmit Client Scripts execute script logic when a form is submitted. Use *onSubmit* Client Scripts to validate field values. For example, if a user submits a Priority 1 record, the script can generate a confirmation dialog notifying the user that the executive staffs are copied on all Priority 1 requests.

User saves, submits, or updates record

onSubmit() Client Scripts execute

Record written to database



Creating Client Scripts

The procedure for adding files to an application in Studio is the same regardless of file type:

1. Click the **Create Application File** link.
2. Choose the new file type, in this case, **Client Script**.
3. Configure the new file.

Configuring the Client Script

As with any script, the configuration tells the script when to execute. The Client Script configuration options are:

- **Name**: Name of Client Script. Use a standard naming scheme to identify custom scripts.
- **Table**: Table to which the script applies.
- **UI Type**: Select whether the script executes for *Desktop and Tablet* or *Mobile/Service Portal* or *All*.
- **Type**: Select when the script runs: *onChange*, *onLoad*, or *onSubmit*.
- **Field Name**: Used only if the script responds to a field value change (*onChange*); name of the field to which the script applies.
- **Active**: Controls whether the script is enabled. Inactive scripts do not execute.
- **Inherited**: If selected, this script applies to the specified table and all tables that inherit from it. For example, a client script on the *Task* table will also apply to the *Change*, *Incident*, *Problem* and all other tables which extend *Task*.
- **Global**: If *Global* is selected the script applies to all Views. If the *Global* field is not selected you must specify the View.
- **View**: Specifies the View to which the script applies. The *View* field is visible when *Global* is not selected. A script can only act on fields that are part of the selected form View. If the *View* field is blank the script applies to the *Default* view.

The screenshot shows the 'Client Script' configuration form in Studio. The form is titled 'Client Script' and 'New record'. It has a 'Submit' button in the top right corner. The form contains the following fields:

- Name**: A text input field with the value 'Needit Set Requested for'.
- Table**: A dropdown menu with the value 'Needit [x_58872_needit_needit]'.
- UI Type**: A dropdown menu with the value 'Desktop'.
- Type**: A dropdown menu with a list of options: '-- None --', 'onCellEdit', 'onChange', 'onLoad' (highlighted), and 'onSubmit'.
- Application**: A dropdown menu with the value 'Needit'.
- Active**: A checkbox that is checked.
- Inherited**: A checkbox that is unchecked.
- Global**: A checkbox that is checked.
- Description**: A text input field that is empty.

The *Field name* field is available for onChange Client Scripts. The *View* field is available when the *Global* option is not selected.

Client Script
New record

Name: NeedIt Set Requested for

Application: NeedIt

Table: NeedIt [x_58872_needit_needit]

Active: ☒

UI Type: Desktop

Inherited: ☐

Type: onChange

Global: ☐

Field name: Active

View: User

Description:

DEVELOPER TIP: Whenever client-side logic is modified, reload the main ServiceNow browser window to ensure the latest logic is loaded.

The Script Field

When the type value is set, a script template is automatically inserted into the *Script* field.

onLoad Script Template


This is the *onLoad* script template:

Script

```
1 function onLoad() {  
2     //Type appropriate comment here, and begin script below  
3  
4 }
```

The *onLoad* function has no arguments passed to it. As indicated by the comment, add your script logic in the *onLoad* function. It is a best practice to document your code so include comments to explain what the script does. Your future self will thank you for the clearly documented script.

This example script generates an alert when a user requests a form load for a record. The user cannot interact with a form until *onLoad* Client Scripts complete execution.

Script ? 

```
1 function onLoad() {
2     //Generate an alert when the form loads
3     alert("Thank you for loading the form. When you close this alert you
4     will be able to interact with the form.");
5 }
```

onSubmit Script Template


This is the *onSubmit* script template:

Script ? 

```
1 function onSubmit() {
2     //Type appropriate comment here, and begin script below
3
4 }
```

The *onSubmit* function has no arguments passed to it. As indicated by the comment, add your script logic in the *onSubmit* function.

This example script generates an alert when a user saves a *NeedIt* record. The record is not submitted to the server until the *onSubmit* Client Scripts complete execution and return true.

Script ? 

```
1 function onSubmit() {
2     //Generate an alert to thank the user for submitting a record
3     alert("Thank you for submitting a record.");
4 }
```

onChange Script Template

This is the *onChange* Script template:

Script ? 

```
1 function onChange(control, oldValue, newValue, isLoading, isTemplate) {
2     if (isLoading || newValue === '') {
3         return;
4     }
5
6     //Type appropriate comment here, and begin script below
7
8 }
```


The *onChange* function is automatically passed five parameters by ServiceNow. Although you do not need to do anything to pass the parameters, you can use them in your script.

- ***control***: field the Client Script is configured for.
- ***oldValue***: value of the field when the form loaded and prior to the change.
- ***newValue***: value of the field after the change.
- ***isLoading***: boolean value indicating whether the change is occurring as part of a form load. Value is *true* if change is due to a form load. When forms load, all the field values on the form change as the record is loaded into the form.
- ***isTemplate***: boolean value indicating whether the change occurred due to population of the field by a template. Value is *true* if change is due to population from a template.

When a user selects a record to load, the form and form layout are rendered first and then the fields are populated with values from the database. From the technical perspective, all field values are changed, from nothing to the record's values, when a form loads. The *if* statement in the template assumes that *onChange* Client scripts should not execute their script logic when a form loads. The *onChange* Client Script also stops execution if the *newValue* for a field is no value. Depending on your use case, you can modify or even remove the *if* statement. For example:

```
//Stop script execution if the field value change was caused  
by a Template
```

```
if(isLoading || newValue === '' || isTemplate) {  
    return;  
}
```

For this example, the *onChange* Client Script executes because of changes to the *Short description* field value:

Type	onChange
Field name	Short description

When a user changes the value of the *Short description* field *on the form* the *onChange* script logic executes. The example generates an alert stating that the value of the *Short description* field has changed from the value the field had when the form loaded to the new value on the form.

Script ?

```
1  function onChange(control, oldValue, newValue, isLoading, isTemplate) {  
2    if (isLoading || newValue === '') {  
3      return;  
4    }  
5  
6    //Generate an alert showing the oldValue and newValue  
7    alert("You have changed the Short description from " + oldValue + " to  
8    " + newValue + ".");  
  }
```

The value of the *Short description* field has changed *only* on the form. Changes are not sent to the database until a user saves, updates, or submits the record. It is important to know that the value of *oldValue* is set *when the form loads*. Regardless of how many times the value of the *Short description* field changes, *oldValue* remains the same until the form is re-loaded from the database.

- 1. **Form loads:**
 - a. *oldValue* = hello
 - b. *newValue* has no value
- 2. **User changes the value in the *Short description* field to byte:**
 - a. *oldValue* = hello
 - b. *newValue* = bye
- 3. **User changes the value in the *Short description* field to greetings:**
 - a. *oldValue* = hello
 - b. *newValue* = greetings
- 4. **User saves the form and reloads the form page:**
 - a. *oldValue* = greetings
 - b. *newValue* has no value

Introduction to Java Script:

An Introduction to JavaScript

Let's see what's so special about JavaScript, what we can achieve with it, and what other technologies play well with it.

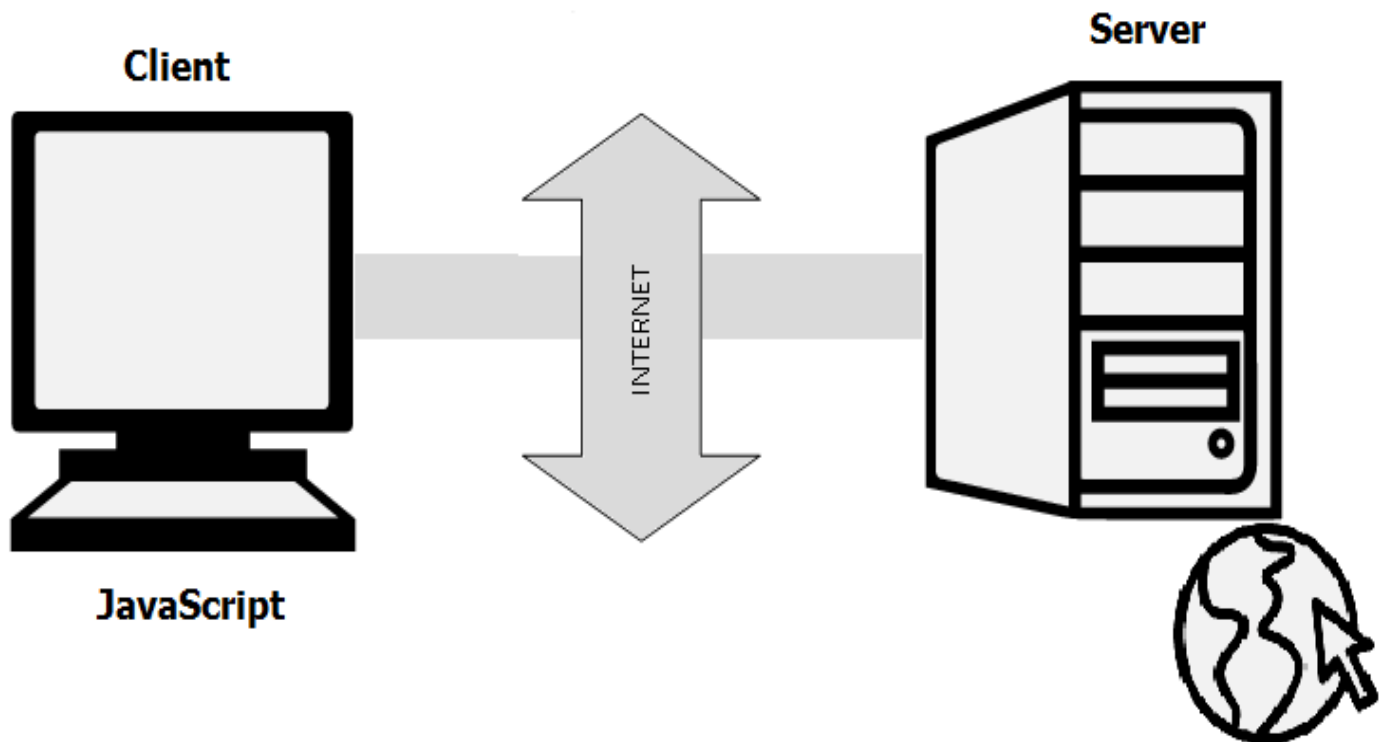
What is JavaScript?

JavaScript is a very powerful **client-side scripting language**. JavaScript is used mainly for enhancing the interaction of a user with the webpage. In other words, you can make your webpage more lively and interactive, with the help of JavaScript. JavaScript is also being used widely in game development and [Mobile](#) application development.

JavaScript was initially created to “make web pages alive”.

The programs in this language are called *scripts*. They can be written right in a web page's HTML and run automatically as the page loads.

Scripts are provided and executed as plain text. They don't need special preparation or compilation to run. In this aspect, JavaScript is very different from another language called [Java](#).



Why is it called JavaScript?

When JavaScript was created, it initially had another name: "LiveScript". But Java was very popular at that time, so it was decided that positioning a new language as a "younger brother" of Java would help. But as it evolved, JavaScript became a fully independent language with its own specification called ECMAScript, and now it has no relation to Java at all.

Today, JavaScript can execute not only in the browser, but also on the server, or actually on any device that has a special program called the JavaScript engine. The browser has an embedded engine sometimes called a "JavaScript virtual machine".

Different engines have different "codenames". For example:

- ❖ **V8** – in Chrome and Opera.
- ❖ **SpiderMonkey** – in Firefox.
- ❖ **There are other codenames like "Chakra" for IE, "ChakraCore" for Microsoft Edge, "Nitro" and "SquirrelFish" for Safari, etc.**

The terms above are good to remember because they are used in developer articles on the internet. We'll use them too. For instance, if "a feature X is supported by V8", then it probably works in Chrome and Opera.

How do engines work?

Engines are complicated. But the basics are easy.

- 1. The engine (embedded if it's a browser) reads ("parses") the script.**
- 2. Then it converts ("compiles") the script to the machine language.**
- 3. And then the machine code runs, pretty fast.**

The engine applies optimizations at each step of the process. It even watches the compiled script as it runs, analyzes the data that flows through it, and further optimizes the machine code based on that knowledge.

JavaScript History:

JavaScript was developed by **Brendan Eich** in 1995, which appeared in Netscape, a popular browser of that time.



**Brendan Eich -
Creator of JavaScript**

How to Run JavaScript?

Being a scripting language, **JavaScript cannot run on its own. In fact, the browser is responsible for running JavaScript code.** When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it is up to the browser to execute it. The main advantage of JavaScript is that **all modern web browsers support JavaScript.** So, you do not have to worry about whether your site visitor uses Internet Explorer, Google Chrome, Firefox or any other browser. JavaScript will be supported. Also, JavaScript **runs on any operating system** including Windows, [Linux](#) or Mac. Thus, JavaScript overcomes the main disadvantages of [VBScript](#) (Now deprecated) which is limited to just IE and Windows.

Tools You Need

To start with, you need a text editor to write your code and a browser to display the web pages you develop.

- ✚ You can use a [text editor](#) of your choice including Notepad++, Visual Studio Code, Sublime Text, Atom or any other text editor you are comfortable with.
- ✚ You can use any [web browser](#) including Google Chrome, Firefox, Microsoft Edge, Internet Explorer etc.

A Simple JavaScript Program

You should place all your JavaScript code within **<script> tags** (<script> and </script>) if you are keeping your JavaScript code within the HTML document itself. This helps your browser distinguish your JavaScript code from the rest of the code. As there are other client-side scripting languages (Example: VBScript), it is highly recommended that you specify the scripting language you use. You have to use the type attribute within the <script> tag and set its value to text/JavaScript like this:

```
<script type="text/javascript">
```

```
<html>
<head>
  <title>My First JavaScript code!!!</title>
  <script>
    alert("Hello World!");
  </script>
</head>
<body>
</body>
</html>
```

What can in-browser JavaScript do?

Modern JavaScript is a “safe” programming language. It does not provide low-level access to memory or CPU, because it was initially created for browsers which do not require it.

JavaScript’s capabilities greatly depend on the environment it’s running in. For instance, [Node.js](#) supports functions that allow JavaScript to read/write arbitrary files, perform network requests, etc. In-browser JavaScript can do everything related to webpage manipulation, interaction with the user, and the web server.

For instance, in-browser JavaScript is able to:

- Add new HTML to the page, change the existing content, and modify styles.
- React to user actions, run on mouse clicks, pointer movements, and key presses.
- Send requests over the network to remote servers download and upload files (so-called [AJAX](#) and [COMET](#) technologies).
- Get and set cookies, ask questions to the visitor, show messages.
- Remember the data on the client-side (“local storage”).

What CAN’T in-browser JavaScript do?

JavaScript’s abilities in the browser are limited for the sake of the user’s safety. The aim is to prevent an evil webpage from accessing private information or harming the user’s data.

Examples of such restrictions include:

- JavaScript on a webpage may not read/write arbitrary files on the hard disk, copy them or execute programs. It has no direct access to OS functions.

Modern browsers allow it to work with files, but the access is limited and only provided if the user does certain actions, like “dropping” a file into a browser window or selecting it via an `<input>` tag.

There are ways to interact with camera/microphone and other devices, but they require a user’s explicit permission. So a JavaScript-enabled page may not sneakily enable a web-camera, observe the surroundings and send the information to the [NSA](#).

- Different tabs/windows generally do not know about each other. Sometimes they do, for example when one window uses JavaScript to open the other one. But even in this case, JavaScript from one page may not access the other if they come from different sites (from a different domain, protocol or port).

This is called the “Same Origin Policy”. To work around that, *both pages* must agree for data exchange and contain a special JavaScript code that handles it. We’ll cover that in the tutorial.

This limitation is, again, for the user’s safety. A page from `http://anysite.com` which a user has opened must not be able to access another browser tab with the URL `http://gmail.com` and steal information from there.

- JavaScript can easily communicate over the net to the server where the current page came from. But its ability to receive data from other sites/domains is crippled. Though possible, it requires

explicit agreement (expressed in HTTP headers) from the remote side. Once again, that's a safety limitation.

What makes JavaScript unique?

There are at least *three* great things about JavaScript:

- Full integration with HTML/CSS.
- Simple things are done simply.
- Support by all major browsers and enabled by default.

JavaScript is the only browser technology that combines these three things.

That's what makes JavaScript unique. That's why it's the most widespread tool for creating browser interfaces. That said; JavaScript also allows creating servers, mobile applications, etc

Some examples of what JavaScript can do.

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is `getElementById()`.

The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":

Example

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

JavaScript accepts both double and single quotes:

Example

```
document.getElementById('demo').innerHTML = 'Hello JavaScript';
```

JavaScript Can Change HTML Attribute Values

In this example JavaScript changes the value of the `src` (source) attribute of an `` tag:

The Light Bulb



Turn on the light

Turn off the light

JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

Example

```
document.getElementById("demo").style.fontSize = "35px";
```

JavaScript Can Hide HTML Elements

Hiding HTML elements can be done by changing the `display` style:

Example

```
document.getElementById("demo").style.display = "none";
```

JavaScript Can Show HTML Elements

Showing hidden HTML elements can also be done by changing the `display` style:

Example

```
document.getElementById("demo").style.display = "block";
```


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 needed.

OR

There are following features of JavaScript:

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
3. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
4. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
5. It is a light-weighted and interpreted language.
6. It is a case-sensitive language.
7. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
8. It provides good control to the users over the web browsers.

Application of JavaScript

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation,
- Dynamic drop-down menus,
- Displaying date and time,
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc.

Platform for 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 in the server-side.
- **Games:** Not only in websites, 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 Ease JS library which provides solutions for working with rich graphics.
- **Smart watches:** JavaScript is being used in all possible devices and applications. It provides a library Pebble JS which is used in smart watch 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, make 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.

Explanation: Some features of JavaScript are as follow:-

1. JavaScript is a **object-based scripting language**.
2. Giving the user more control over the browser.
3. It Handling dates and time.
4. It Detecting the user's browser and OS,
5. It is light weighted.
6. JavaScript is a scripting language and it is not java.
7. JavaScript is interpreter based scripting language.
8. JavaScript is case sensitive.

JavaScript Types:

JavaScript Data Types

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

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

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

1. `var a=40;` holding number
2. `var b="Rahul";` holding string

JavaScript primitive data types:

There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JavaScript non-primitive data types:

The non-primitive data types are as follows:

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

JavaScript Data Types:

JavaScript variables can hold many **data types**: numbers, strings, objects and more:

```
var length = 16;           // Number
var lastName = "Johnson";  // String
var x = {firstName:"John", lastName:"Doe"}; // Object
```

The Concept of Data Types:

In programming, data types are 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:

```
var 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:

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

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

Example:

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

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

JavaScript Types are Dynamic:

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

Example

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

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

```
var carName1 = "Volvo XC60"; // Using double quotes
var carName2 = 'Volvo XC60'; // Using single quotes
```

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

Example

```
var answer1 = "It's alright"; // Single quote inside double quotes
var answer2 = "He is called 'Johnny'"; // Single quotes inside double quotes
var answer3 = 'He is called "Johnny"'; // Double quotes inside single quotes
```

JavaScript Numbers:

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

Example

```
var x1 = 34.00;    // Written with decimals
var x2 = 34;       // Written without decimals
```

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

Example

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

JavaScript Booleans:

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

Example

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

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

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

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

JavaScript Objects:

JavaScript objects are written with curly braces `{}`.

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

Example

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

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

Undefined:

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

Example

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

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
```

Empty Values:

An empty value has nothing to do with `undefined`.

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

Example

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

Null:

In JavaScript `null` is "nothing". It is supposed to be something that doesn't exist.

Unfortunately, in JavaScript, the data type of `null` is an object.

You can consider it a bug in JavaScript that `typeof null` is an object. It should be `null`.

You can empty an object by setting it to `null`:

Example

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = null;    // Now value is null, but type is still an object
```

Variables in JavaScript:

A **JavaScript variable** is simply a name of storage location.

There are two types of variables in JavaScript:

- Local variable
- Global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore (_), or dollar (\$) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables

Example of JavaScript variable

Let's see a simple example of JavaScript variable.

<script>

```
var x = 10;
```

```
var y = 20;
```

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

```
document.write(z);
```

</script>

Output of the above example

30

JavaScript local variable:

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

```
<script>
```

```
function abc(){  
var x=10;//local variable  
}
```

```
</script>
```

Or,

```
<script>
```

```
If(10<13){  
var y=20;//JavaScript local variable  
}
```

```
</script>
```

JavaScript global variable:

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

For example:

```
<script>
```

```
var data=200;//global variable  
function a(){  
document.writeln(data);  
}  
function b(){  
document.writeln(data);  
}  
a();//calling JavaScript function  
b();
```

```
</script>
```

Operators in JS:

JavaScript operators are symbols that are used to perform operations on operands. For example:

```
var sum=10+20;
```

Here, + is the arithmetic operator and = is the assignment operator.

Example:

Assign values to variables and add them together:

```
var x = 5;           // assign the value 5 to x
var y = 2;           // assign the value 2 to y
var z = x + y;       // assign the value 7 to z (5 + 2)
```

There are following types of operators in JavaScript.

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

1. The assignment operator (=) assigns a value to a variable.

```
var x = 10;
```

2. The addition operator (+) adds numbers:

```
var x = 5;
var y = 2;
var z = x + y;
```

3. The multiplication operator (*) multiplies numbers.

```
var x = 5;
var y = 2;
var z = x * y;
```

JavaScript Arithmetic Operators:

Arithmetic operators are used to perform arithmetic on numbers:

Operator	Description
<code>+</code>	Addition
<code>-</code>	Subtraction
<code>*</code>	Multiplication
<code>**</code>	Exponentiation (ES2016)
<code>/</code>	Division
<code>%</code>	Modulus (Division Remainder)
<code>++</code>	Increment
<code>--</code>	Decrement

JavaScript Assignment Operators:

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
<code>=</code>	<code>x = y</code>	<code>x = y</code>
<code>+=</code>	<code>x += y</code>	<code>x = x + y</code>
<code>-=</code>	<code>x -= y</code>	<code>x = x - y</code>
<code>*=</code>	<code>x *= y</code>	<code>x = x * y</code>
<code>/=</code>	<code>x /= y</code>	<code>x = x / y</code>
<code>%=</code>	<code>x %= y</code>	<code>x = x % y</code>
<code>**=</code>	<code>x **= y</code>	<code>x = x ** y</code>

JavaScript String Operators:

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

```
var txt1 = "John";  
var txt2 = "Doe";  
var txt3 = txt1 + " " + txt2;
```

The result of txt3 will be:

John Doe

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

```
var txt1 = "What a very ";  
txt1 += "nice day";
```

The result of txt1 will be:

What a very nice day

Adding Strings and Numbers:

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

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

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

10
55
Hello5

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

JavaScript Logical Operators:

Operator	Description
&&	logical and
	logical or
!	logical not

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

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
<<	Zero fill left shift	5 << 1	0101 << 1	1010	10
>>	Signed right shift	5 >> 1	0101 >> 1	0010	2
>>>	Zero fill 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.

~00000000000000000000000000000000101 will return

11111111111111111111111111111111010

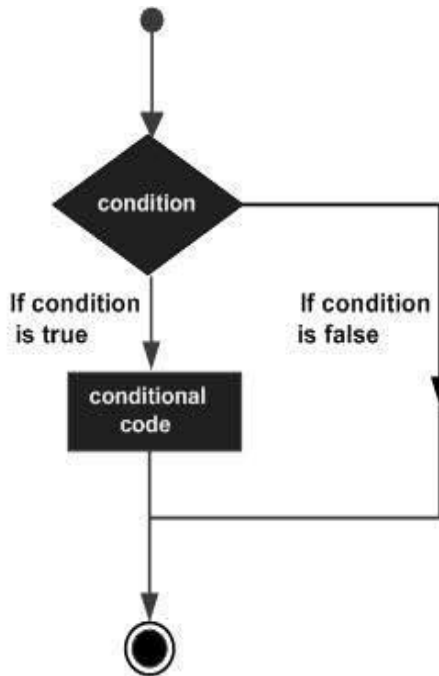
Conditions Statements:

While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.

JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the **if..else** statement.

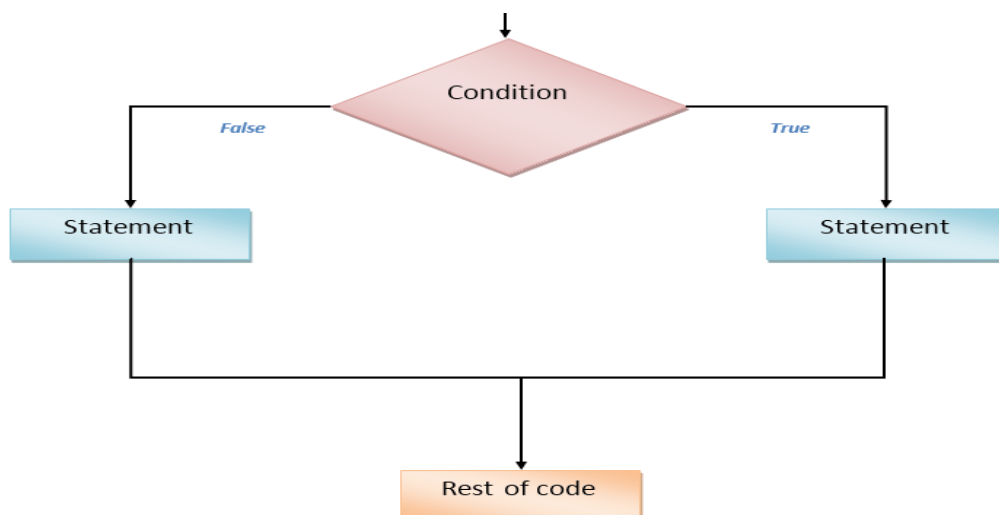
Flow Chart of if-else:

The following flow chart shows how the if-else statement works.



JavaScript supports the following forms of if..Else statement –

- if statement
- if...else statement
- if...else if... statement.



Definition and Usage:

The if/else statement executes a block of code if a specified condition is true. If the condition is false, another block of code can be executed. The if/else statement is a part of JavaScript's "Conditional" Statements, which are used to perform different actions based on different conditions.

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 select one of many blocks of code to be executed.

Syntax

- **The if statement specifies a block of code to be executed if a condition is true:**

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

- **The else statement specifies 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  
}
```

- **The else if statement specifies a new condition if the first condition is false:**

```
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  
}
```

Parameter Values

Parameter	Description
<i>condition</i>	Required. An expression that evaluates to true or false

Example:

If the time is less than 20:00, create a "Good day" greeting, otherwise "Good evening":

```
var time = new Date().getHours();
if (time < 20) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

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":

```
var time = new Date().getHours();
if (time < 10) {
    greeting = "Good morning";
} else if (time < 20) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

Example:

If the first <div> element in the document has an id of "myDIV", change its font-size:

```
var x = document.getElementsByTagName("DIV")[0];

if (x.id === "myDIV") {
    x.style.fontSize = "30px";
}
```

Change the value of the source attribute (src) of an element, if the user clicks on the image:

```

```

```
<script>
function changeImage() {
    var image = document.getElementById("myImage");
    if (image.src.match("bulbon")) {
```

```
    image.src = "pic_bulboff.gif";  
  } else {  
    image.src = "pic_bulbon.gif";  
  }  
}  
</script>
```

JavaScript Loop:

Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true. For example, suppose we want to print “Hello World” 10 times. This can be done in two ways as shown below:

Iterative Method

Iterative method to do this is to write the document.write() statement 10 times.

```
<script type = "text/javascript">  
document.write("Hello World\n");  
document.write("Hello World\n");  
document.write("Hello World\n");  
document.write("Hello World\n");  
document.write("Hello World\n");  
document.write("Hello World\n");  
document.write("Hello World\n");  
document.write("Hello World\n");  
document.write("Hello World\n");  
document.write("Hello World\n");  
< /script>
```

In Loop, the statement needs to be written only once and the loop will be executed 10 times as shown below:

```
<script type = "text/javascript">  
var i;  
for (i = 0; i < 10; i++)  
{  
    document.write("Hello World!\n");  
}  
< /script>
```

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

There are four types of loops in JavaScript.

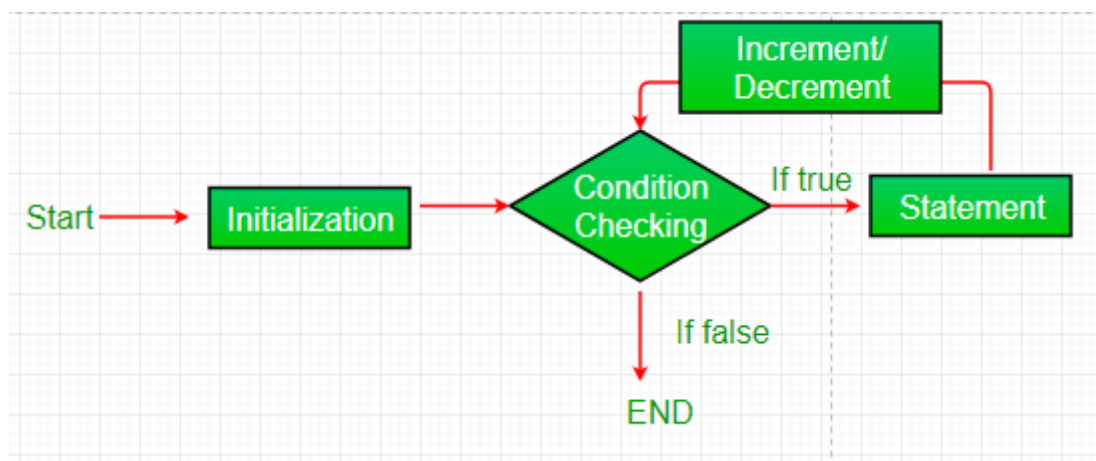
1. for loop
2. while loop
3. do-while loop
4. for-in loop

1) JavaScript For loop:

- ❖ The **JavaScript for loop** iterates the elements for the fixed number of times. It should be used if number of iteration is known. The syntax of for loop is given below.
- ❖ **for loop:** for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

```
for (initialization condition; testing  
condition;  
  
increment/decrement)  
{  
    statement(s)  
}
```

Flowchart:



1. **Initialization condition:** Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.
2. **Testing Condition:** It is used for testing the exit condition for a loop. It must return a boolean value. It is also an **Entry Control Loop** as the condition is checked prior to the execution of the loop statements.
3. **Statement execution:** Once the condition is evaluated to true, the statements in the loop body are executed.
4. **Increment/ Decrement:** It is used for updating the variable for next iteration.
5. **Loop termination:** When the condition becomes false, the loop terminates marking the end of its life cycle.

```
<script type = "text/javascript">

// JavaScript program to illustrate for loop

var x;

// for loop begins when x=2
// and runs till x <=4
for (x = 2; x <= 4; x++)
{
    document.write("Value of x:" + x + "<br />");
}

< /script>
```

Output:

```
Value of x:2
Value of x:3
Value of x:4
```

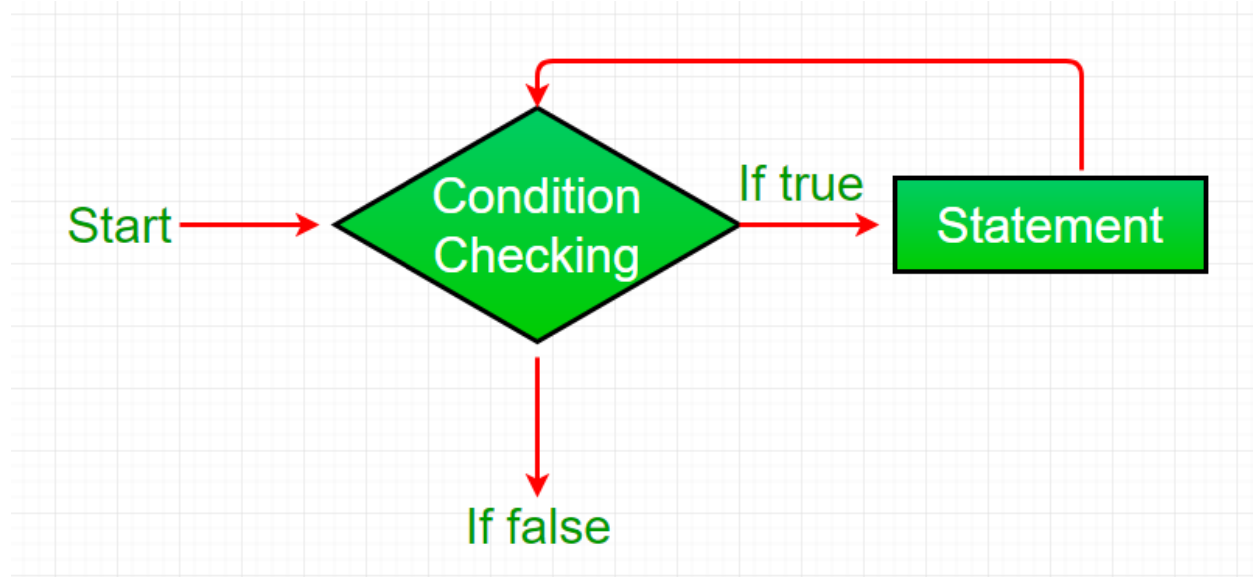
2) JavaScript while loop:

- ❖ The **JavaScript while loop** iterates the elements for the infinite number of times. It should be used if number of iteration is not known. The syntax of while loop is given below.
- ❖ **while loop:** A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax :

```
while (boolean condition)
{
    loop statements...
}
```

Flowchart:



- While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called **Entry control loop**
- Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

```
<script type = "text/javascript">

// JavaScript program to illustrate while loop

var x = 1;

// Exit when x becomes greater than 4
while (x <= 4)
{
    document.write("Value of x:" + x + "<br />");

    // increment the value of x for
    // next iteration
    x++;
}

< /script>
```


Output:

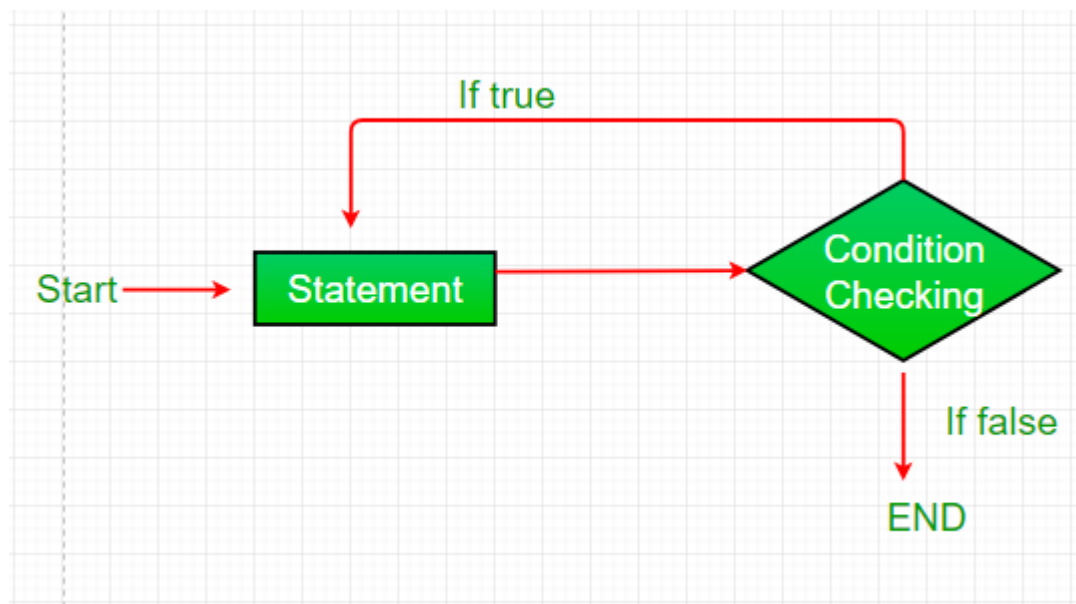
```
Value of x:1  
Value of x:2  
Value of x:3  
Value of x:4
```

3) JavaScript do while loop:

- ❖ The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least* once whether condition is true or false. The syntax of do while loop is given below.
- ❖ **do while:** do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of Exit Control Loop.

Syntax:

```
do  
{  
    statements..  
}  
while (condition);
```



1. do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
2. After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.
3. When the condition becomes false, the loop terminates which marks the end of its life cycle.
4. It is important to note that the do-while loop will execute its statements at-least once before any condition is checked, and therefore is an example of exit control loop.

```
<script type = "text/javascript">  
  
// JavaScript program to illustrate do-while loop  
  
var x = 21;  
  
do  
{  
    // The line while be printer even  
    // if the condition is false  
    document.write("Value of x:" + x + "<br />");  
  
    x++;  
} while (x < 20);  
  
< /script>
```

Output:

Value of x: 21

JS Popup Boxes:

In JavaScript, popup boxes are used to display the message or notification to the user.

There are three types of pop up boxes in JavaScript namely

1. **Alert Box**
2. **Confirm Box**
3. **Prompt Box.**

1. Alert Box:

An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires entering some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed.

Syntax:

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

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

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function Warn() {
          alert ("This is a warning message!");
          document.write ("This is a warning message!");
        }
      //-->
    </script>
  </head>

  <body>
    <p>Click the following button to see the result: </p>
    <form>
      <input type = "button" value = "Click Me" onclick = "Warn();"
/>
    </form>
  </body>
</html>
```

Output:

This is a warning message!

2. Confirm Box:

A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: **OK** and **Cancel**.

If the user clicks on the OK button, the window method **confirm ()** will return true. If the user clicks on the Cancel button, then **confirm ()** returns false. You can use a confirmation dialog box as follows.

Syntax:

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

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

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function getConfirmation() {
          var retVal = confirm("Do you want to continue ?");
          if( retVal == true ) {
            document.write ("User wants to continue!");
            return true;
          } else {
            document.write ("User does not want to continue!");
            return false;
          }
        }
      //-->
    </script>
  </head>

  <body>
    <p>Click the following button to see the result: </p>
    <form>
      <input type = "button" value = "Click Me" onclick =
"getConfirmation();" />
    </form>
  </body>
</html>
```

Output:

User wants to continue!

3. Prompt Box:

The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

This dialog box is displayed using a method called `prompt()` which takes two parameters:

- (i) a label which you want to display in the text box and
- (ii) a default string to display in the text box.

This dialog box has two buttons: **OK** and **Cancel**. If the user clicks the OK button, the window method **`prompt()`** will return the entered value from the text box. If the user clicks the Cancel button, the window method **`prompt()`** returns **`null`**.

Syntax:

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

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

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function getValue() {
          var retVal = prompt("Enter your name : ", "your name
here");
          document.write("You have entered : " + retVal);
        }
      //-->
    </script>
  </head>

  <body>
    <p>Click the following button to see the result: </p>
    <form>
      <input type = "button" value = "Click Me" onclick =
"getValue();" />
    </form>
  </body>
</html>
```

Output:

You have entered: your name here

JS Events:

What is an Event?

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

JavaScript Events:

The change in the state of an object is known as an **Event**. In html, there are various events which represents that some activity is performed by the user or by the browser. When [javascript](#) code is included in [HTML](#), js react over these events and allow the execution. This process of reacting over the events is called **Event Handling**. Thus, js handles the HTML events via **Event Handlers**.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

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:

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

Some of the HTML events and their event handlers are:

➤ Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

➤ Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

➤ Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

➤ Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Let's discuss some examples over events and their handlers.

Click Event:

```
<html>
<head> Javascript Events </head>
<body>
<script language="Javascript" type="text/Javascript">
  <!--
  function clickevent()
  {
    document.write("This is JavaTpoint");
  }
  //-->
</script>
<form>
<input type="button" onclick="clickevent()" value="Who's this?"/>
</form>
</body>
</html>
```


Mouse over Event:

```
<html>
<head>
<h1> Javascript Events </h1>
</head>
<body>
<script language="Javascript" type="text/Javascript">
  <!--
  function mouseoverevent()
  {
    alert("This is JavaTpoint");
  }
  //-->
</script>
<p onmouseover="mouseoverevent()"> Keep cursor over me</p>
</body>
</html>
```

Focus Event:

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onfocus="focusevent()" />
<script>
<!--
  function focusevent()
  {
    document.getElementById("input1").style.background=" aqua";
  }
  //-->
</script>
</body>
</html>
```

Key-down Event

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onkeydown="keydownevent()"/>
<script>
<!--
  function keydownevent()
  {
    document.getElementById("input1");
    alert("Pressed a key");
  }
//-->
</script>
</body>
</html>
```

Load event:

```
<html>
<head> Javascript Events</head>
</br>
<body onload="window.alert('Page successfully loaded');">
<script>
<!--
document.write("The page is loaded successfully");
//-->
</script>
</body>
</html>
```

JS Arrays, Working with Arrays:

Arrays in JavaScript:

✚ In JavaScript, array is a single variable that is used to store different elements. It is often used when we want to store list of elements and access them by a single variable. Unlike most languages where array is a reference to the multiple variable, in JavaScript array is a single variable that stores multiple elements.

✚ JavaScript arrays are used to store multiple values in a single variable.

Example:

```
var cars = ["Saab" "Volvo""BMW"];
```

What is an Array?

An array is a special variable, which can hold more than one value at a time.

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

```
var car1 = "Saab";  
var car2 = "Volvo";  
var 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.

There are 3 ways to construct array in JavaScript:

1. By array literal
2. By creating instance of Array directly (using new keyword)
3. By using an Array constructor (using new keyword)

Creating an Array:

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

Syntax:

```
var array_name = [item1 item2 ...];
```

Example:

```
var cars = ["Saab" "Volvo" "BMW"];
```

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

Example:

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

Using the JavaScript Keyword new

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

Example:

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

The two examples above do exactly the same. There is no need to use new Array().

For simplicity, readability and execution speed, use the first one (the array literal method).

1. Access the Full Array:

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

Example:

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

2. 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:

```
var person = ["John" "Doe" 46];
```

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

Object:

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

3. Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

Examples:

```
var x = cars.length;    // The length property returns the number of elements  
var y = cars.sort();    // The sort() method sorts arrays
```

The length Property:

The **length** property of an array returns the length of an array (the number of array elements).

Example:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.length;    // the length of fruits is 4
```

JS Objects :

JavaScript Objects:

A JavaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript. JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

In JavaScript, objects are king. If you understand objects, you understand JavaScript.

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the **new** keyword)
- Numbers can be objects (if defined with the **new** keyword)
- Strings can be objects (if defined with the **new** keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects



Objects in Javascript

Last Updated: 29-06-2020

Objects, in JavaScript, is it's most important data-type and forms the building blocks for modern JavaScript. These objects are quite different from JavaScript's primitive data-types (Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).

- Objects are more complex and each object may contain any combination of these primitive data-types as well as reference data-types.
- An object, is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don't actually store the value.
- Loosely speaking, **objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of "key: value" pairs**. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.

An object can be created with figure brackets {...} with an optional list of properties. A property is a "key: value" pair, where a key is a string (also called a "property name"), and value can be anything.

To understand this rather abstract definition, let us look at an example of a JavaScript Object :

```
let school = {  
  name : "Vivekananda School",  
  location : "Delhi",  
  established : "1971"  
}
```

In the above example **"name", "location", "established"** are all **"keys"** and **"Vivekananda School", "Delhi" and 1971** are values of these keys respectively.

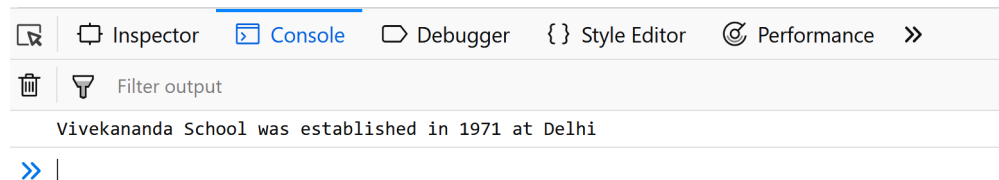
Each of these keys is referred to as **properties** of the object. An object in JavaScript may also have a function as a member, in which case it will be known as a **method** of that object.

Let us see such an example :



```
// javascript code demonstrating a simple object
let school = {
  name: 'Vivekananda School',
  location : 'Delhi',
  established : '1971',
  displayInfo : function(){
    console.log(`${school.name} was established
      in ${school.established} at ${school.location}`);
  }
}
school.displayInfo();
```

Output:



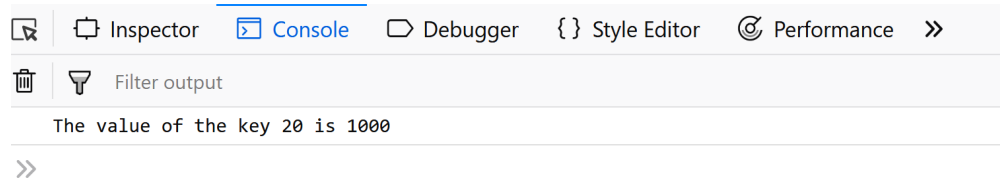
In the above example, **"displayinfo"** is a method of the school object that is being used to work with the object's data, stored in its properties.



must be accessed using the "bracket notation" like this :

```
let school = {  
  name: 'Vivekananda School',  
  location : 'Delhi',  
  established : '1971',  
  20 : 1000,  
  displayInfo : function(){  
    console.log(`The value of the key 20 is ${school['20']}`);  
  }  
}  
school.displayInfo();
```

Output:



But more on the bracket notation later.

Property names can also be strings with more than one space separated words. In which case, these property names must be enclosed in quotes :

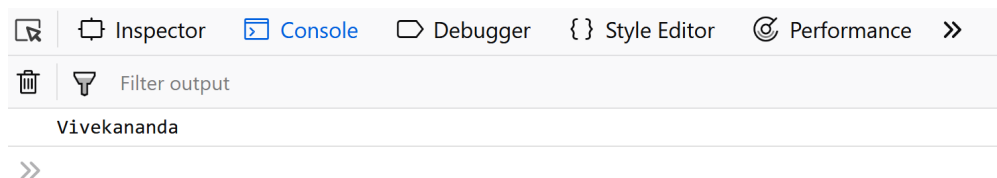
```
let school = {  
  "school name" : "Vivekananda School",  
}
```

Like property names which are numbers, they must also be accessed using the bracket notation. Like if we want to access the 'Vivekananda' from 'Vivekananda School' we can do something like this:



```
// bracket notation
let school = {
  name: 'Vivekananda School',
  displayInfo : function(){
    console.log(`${school.name.split(' ')[0]}`);
  }
}
school.displayInfo(); // Vivekananda
```

Output:



In the above code, we made use of bracket notation and also split method provided by javascript which you will learn about in strings article.

Inherited Properties



property. To verify if a property is an object's own property, we can use the `hasOwnProperty` method.

Property Attributes

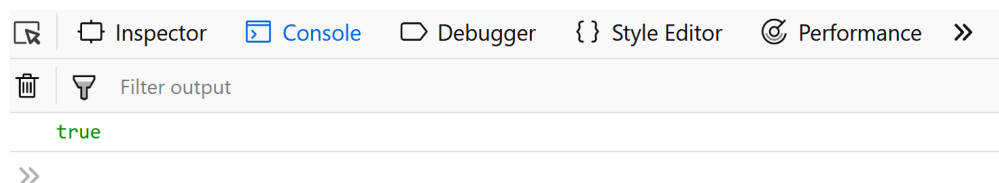
Data properties in JavaScript have four attributes.

- **value:** The property's value.
- **writable:** When true, the property's value can be changed
- **enumerable:** When true, the property can be iterated over by "for-in" enumeration. Otherwise, the property is said to be non-enumerable.
- **configurable:** If false, attempts to delete the property, change the property to be an access-or property, or change its attributes (other than `[[Value]]`, or changing `[[Writable]]` to false) will fail.

```
// hasOwnProperty code in js
const object1 = new Object();
object1.property1 = 42;

console.log(object1.hasOwnProperty('property1')); // true
```

Output:



Creating Objects

There are several ways or syntax's to create objects. One of which, known as the Object literal syntax, we have already used. Besides the object literal syntax, objects in JavaScript may also be created using the constructors, Object Constructor or the prototype pattern.



```
var obj = {  
  member1 : value1,  
  member2 : value2,  
};
```

These members can be anything – strings, numbers, functions, arrays or even other objects. An object like this is referred to as an object literal. This is different from other methods of object creation which involve using constructors and classes or prototypes, which have been discussed below.

2. **Object Constructor** : Another way to create objects in JavaScript involves using the “Object” constructor. The Object constructor creates an object wrapper for the given value. This, used in conjunction with the “new” keyword allows us to initialize new objects. Example :

```
const school = new Object();  
school.name = 'Vivekanada school';  
school.location = 'Delhi';  
school.established = 1971;  
  
school.displayInfo = function(){  
  console.log(`${school.name} was established  
    in ${school.established} at ${school.location}`);  
}  
  
school.displayInfo();
```



```
Vivekananda School was established in 1971 at Delhi
```

```
>> |
```

Artiva

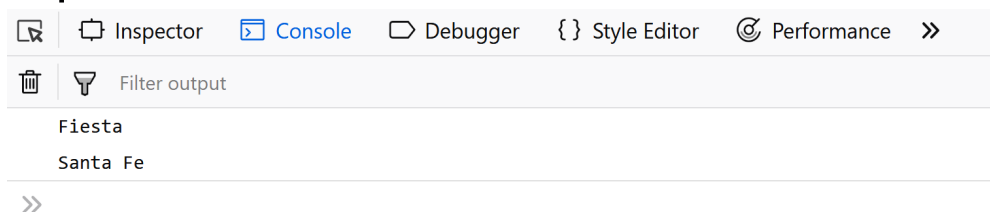
The two methods mentioned above are not well suited to programs that require the creation of multiple objects of the same kind, as it would involve repeatedly writing the above lines of code for each such object. To deal with this problem, we can make use of two other methods of object creation in JavaScript that reduces this burden significantly, as mentioned below:

3. **Constructors:** Constructors in JavaScript, like in most other OOP languages, provides a template for creation of objects. In other words, it defines a set of properties and methods that would be common to all objects initialized using the constructor.

Let us see an example :

```
function Vehicle(name, maker) {  
    this.name = name;  
    this.maker = maker;  
}  
  
let car1 = new Vehicle('Fiesta', 'Ford');  
let car2 = new Vehicle('Santa Fe', 'Hyundai')  
  
console.log(car1.name);    // Output: Fiesta  
console.log(car2.name);    // Output: Santa Fe
```

Output:



Notice the usage of the “new” keyword before the function Vehicle. Using the “new” keyword in this manner before any function turns it into a constructor. What the “new Vehicle()” actually does is :

- It creates a new object and sets the constructor property of the object to schools (It is important to note that this property is a special default property that is not



function in JavaScript gets a prototype object, which is initially just an empty object but can be modified. The object, when instantiated inherits all properties from its constructor's prototype object).

- Then calls `Vehicle()` in the context of the new object, which means that when the "this" keyword is encountered in the constructor(`vehicle()`), it refers to the new object that was created in the first step.
- Once this is finished, the newly created object is returned to `car1` and `car2` (in the above example).

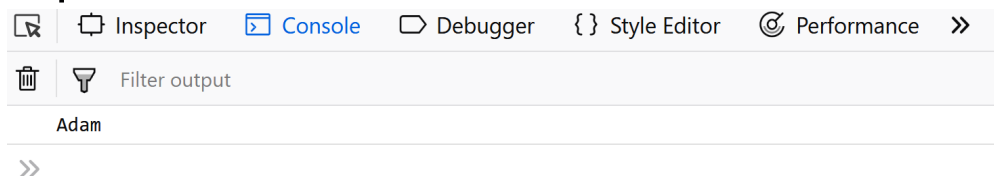
Inside classes, there can be special methods named `constructor()`.

```
class people {  
  constructor()  
  {  
    this.name = "Adam";  
  }  
}
```

```
let person1 = new people();
```

```
// Output : Adam  
console.log(person1.name);
```

Output:



Having more than one function in a class with the name of `constructor()` results in an error.

4. **Prototypes** : Another way to create objects involves using prototypes. Every JavaScript function has a `prototype` object property by default (it is empty by default). Methods or properties may be attached to this property. A detailed description of prototypes is beyond the scope of this introduction to objects.

However you may familiarize yourself with the basic syntax used as below:

```
let obj = Object.create(prototype_object, propertiesObject)  
// the second propertiesObject argument is optional
```

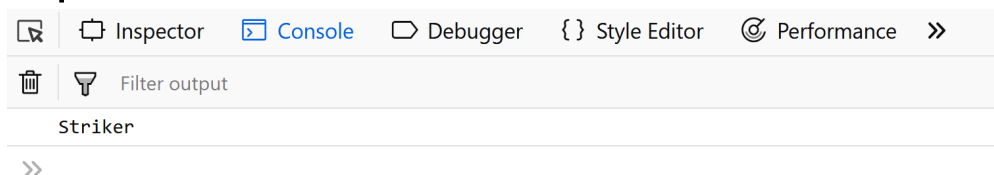
An example of making use of the `Object.create()` method is:



```
let footballer1 = Object.create(footballers);
```

```
// Output : Striker  
console.log(footballer1.position);
```

Output:



In the above example footballers served as a prototype for creating the object "footballer1".

All objects created in this way inherits all properties and methods from its prototype objects. Prototypes can have prototypes and those can have prototypes and so on. This is referred to as prototype chaining in JavaScript. This chain terminates with the `Object.prototype` which is the default prototype fallback for all objects. Javascript objects, by default, inherit properties and methods from `Object.prototype` but these may easily be overridden. It is also interesting to note that the default prototype is not always `Object.prototype`. For example Strings and Arrays have their own default prototypes – `String.prototype` and `Array.prototype` respectively.

Accessing Object Members

Object members(properties or methods) can be accessed using the :

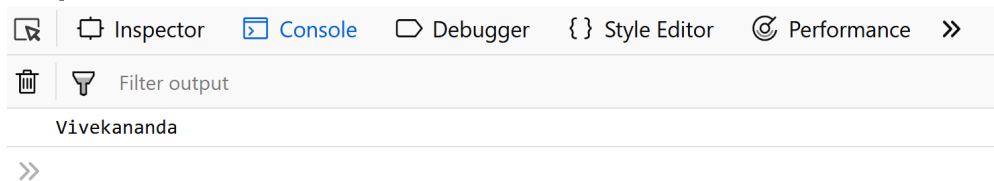


```
let school = {
  name : "Vivekanada",
  location : "Delhi",
  established : 1971,
  20 : 1000,
  displayinfo : function() {
    console.log(`${school.name} was established
      in ${school.established} at ${school.location}`);
  }
}

console.log(school.name);

console.log(school.established);
```

Output:



2. Bracket Notation :

```
objectName["memberName"]
```

```
let school = {
  name : "Vivekanada School",
  location : "Delhi",
  established : 1995,
  20 : 1000,
  displayinfo : function() {
    document.write(`${school.name} was established
      in ${school.established} at ${school.location}`);
  }
}

// Output : Vivekanada School
console.log(school['name']);

// Output: 1000
console.log(school['20']);
```




Filter output
Vivekanada School
1000
>>

Unlike the dot notation, the bracket keyword works with any string combination, including, but not limited to multi-word strings.

For example:

```
somePerson.first name // invalid
somePerson["first name"] // valid
```

Unlike the dot notation, the bracket notation can also contain names which are results of any expressions variables whose values are computed at run-time.

For instance :

```
let key = "first name" somePerson[key] = "Name Surname"
```

Similar operations are not possible while using the dot notation.

Iterating over all keys of an object

To iterate over all existing enumerable keys of an object, we may use the `for...in` construct. It is worth noting that this allows us to access only those properties of an object which are enumerable (Recall that enumerable is one of the four attributes of data properties).

For instance, properties inherited from the `Object.prototype` are not enumerable. But, enumerable properties inherited from somewhere can also be accessed using the `for...in` construct

Example:

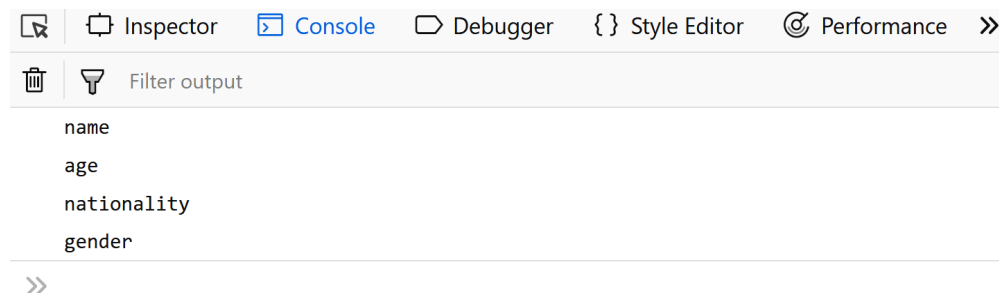
```
let person = {
  gender : "male"
}

var person1 = Object.create(person);
person1.name = "Adam";
person1.age = 45;
person1.nationality = "Australian";

for (let key in person1) {
  // Output : name, age, nationality
}
```



Output:

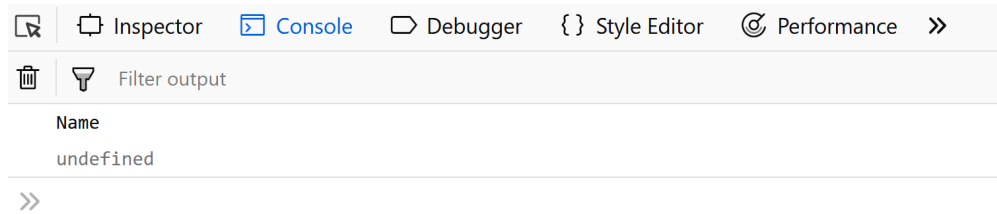


Deleting Properties

To Delete a property of an object we can make use of the `delete` operator. An example of its usage has been listed below:

```
let obj1 = {  
  propfirst : "Name"  
}  
  
// Output : Name  
console.log(obj1.propfirst);  
delete obj1.propfirst  
  
// Output : undefined  
console.log(obj1.propfirst);
```

Output:



It is important to note that we can not delete inherited properties or non-configurable properties in this manner.

For example :

```
let obj1 = {
  propfirst : "Name"
}
// Output : Name
console.log(obj1.propfirst)
let obj2 = Object.create(obj1);

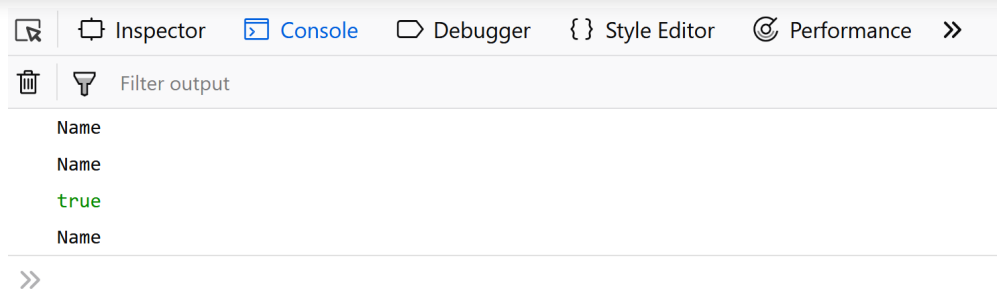
// Output : Name
console.log(obj2.propfirst);

// Output : true.
console.log(delete obj2.propfirst);

// Surprisingly Note that this will return true
// regardless of whether the deletion was successful

// Output : Name
console.log(obj2.propfirst);
```

Output:



GeeksforGeeks

~~₹17,999~~
₹10,999

Register Now

**FULL STACK
DEVELOPMENT**
LIVE

Recommended Posts:

Extract unique objects by attribute from array of objects.

[JavaScript Course | Objects in JavaScript](#)

[How to compare two JavaScript array objects using jQuery/JavaScript ?](#)

[How to merge properties of two JavaScript objects dynamically?](#)



Sort an array of objects using Boolean property in JavaScript

Sort array of objects by string property value in JavaScript

Max/Min value of an attribute in an array of objects in JavaScript

JavaScript | Date Objects

How to remove duplicates from an array of objects using JavaScript ?

How to remove Objects from Associative Array in JavaScript ?

How to Sort/Order keys in JavaScript objects ?

How to remove object from array of objects using JavaScript ?

How to implement a filter() for Objects in JavaScript?

How to convert JSON string to array of JSON objects using JavaScript ?

JavaScript | window.location and document.location Objects

Equality for two JavaScript objects

How to check two objects have same data using JavaScript ?

How to filter nested objects in JavaScript ?



Daipayan

Check out this Author's [contributed articles](#).

If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please Improve this article if you find anything incorrect by clicking on the "Improve Article" button below.

Improved By : [immukul](#)

Article Tags : [JavaScript](#) [Web Technologies](#) [javascript-basics](#) [javascript-oop](#)



2.3

☐ To-do ☐ DoneBased on **3** vote(s)[Improve Article](#)

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

 5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

 feedback@geeksforgeeks.org

Company

[About Us](#)
[Careers](#)
[Privacy Policy](#)
[Contact Us](#)

Practice

Learn

[Algorithms](#)
[Data Structures](#)
[Languages](#)
[CS Subjects](#)
[Video Tutorials](#)

Contribute



How to begin?

Videos

@geeksforgeeks , Some rights reserved

JavaScript Objects

A JavaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don't create class to get the object. But, we directly create objects.

Creating Objects in JavaScript

There are 3 ways to create objects.

1. By object literal
2. By creating instance of Object directly (using new keyword)
3. By using an object constructor (using new keyword)

1) JavaScript Object by object literal

The syntax of creating object using object literal is given below:

```
object={property1:value1,property2:value2.....propertyN:valueN}
```

As you can see, property and value is separated by : (colon).

Let's see the simple example of creating object in JavaScript.

```
<script>
```




```
emp={id:102,name:"Shyam Kumar",salary:40000}  
document.write(emp.id+" "+emp.name+" "+emp.salary);  
</script>
```

[Test it Now](#)

Output of the above example

102 Shyam Kumar 40000



2) By creating instance of Object

The syntax of creating object directly is given below:

```
var objectname=new Object();
```

Here, **new keyword** is used to create object.

Let's see the example of creating object directly.

```
<script>  
var emp=new Object();  
emp.id=101;  
emp.name="Ravi Malik";  
emp.salary=50000;  
document.write(emp.id+" "+emp.name+" "+emp.salary);  
</script>
```

[Test it Now](#)

Output of the above example

101 Ravi 50000

3) By using an Object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

The **this keyword** refers to the current object.

The example of creating object by object constructor is given below.



```
<script>
function emp(id,name,salary){
  this.id=id;
  this.name=name;
  this.salary=salary;
}
e=new emp(103,"Vimal Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);
</script>
```

[Test it Now](#)

Output of the above example

```
103 Vimal Jaiswal 30000
```

Defining method in JavaScript Object

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

The example of defining method in object is given below.

```
<script>
function emp(id,name,salary){
  this.id=id;
  this.name=name;
  this.salary=salary;

  this.changeSalary=changeSalary;
  function changeSalary(otherSalary){
    this.salary=otherSalary;
  }
}
e=new emp(103,"Sonoo Jaiswal",30000);
document.write(e.id+" "+e.name+" "+e.salary);
e.changeSalary(45000);
document.write("<br>" +e.id+" "+e.name+" "+e.salary);
</script>
```

[Test it Now](#)

Output of the above example

```
103 Sonoo Jaiswal 30000
103 Sonoo Jaiswal 45000
```

JavaScript Object Methods

The various methods of Object are as follows:



S.No	Methods	Description
1	<code>Object.assign()</code>	This method is used to copy enumerable and own properties from a source object to a target object
2	<code>Object.create()</code>	This method is used to create a new object with the specified prototype object and properties.
3	<code>Object.defineProperty()</code>	This method is used to describe some behavioral attributes of the property.
4	<code>Object.defineProperties()</code>	This method is used to create or configure multiple object properties.
5	<code>Object.entries()</code>	This method returns an array with arrays of the key, value pairs.
6	<code>Object.freeze()</code>	This method prevents existing properties from being removed.
7	<code>Object.getOwnPropertyDescriptor()</code>	This method returns a property descriptor for the specified property of the specified object.
8	<code>Object.getOwnPropertyDescriptors()</code>	This method returns all own property descriptors of a given object.
9	<code>Object.getOwnPropertyNames()</code>	This method returns an array of all properties (enumerable or not) found.
10	<code>Object.getOwnPropertySymbols()</code>	This method returns an array of all own symbol key properties.
11	<code>Object.getPrototypeOf()</code>	This method returns the prototype of the specified object.
12	<code>Object.is()</code>	This method determines whether two values are the same value.
13	<code>Object.isExtensible()</code>	This method determines if an object is extensible
14	<code>Object.isFrozen()</code>	This method determines if an object was frozen.
15	<code>Object.isSealed()</code>	This method determines if an object is sealed.
16	<code>Object.keys()</code>	This method returns an array of a given object's own property names.
17	<code>Object.preventExtensions()</code>	This method is used to prevent any extensions of an object.
18	<code>Object.seal()</code>	This method prevents new properties from being added and marks all existing properties as non-configurable.
19	<code>Object.setPrototypeOf()</code>	This method sets the prototype of a specified object to another object.
20	<code>Object.values()</code>	This method returns an array of values.

[< prev](#)
[next >](#)

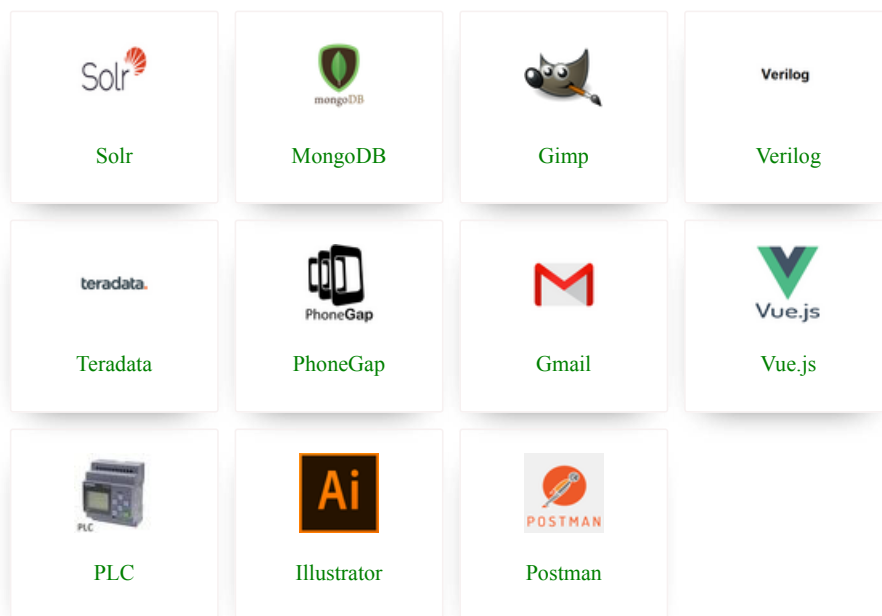

For Videos Join Our Youtube Channel: [Join Now](#)

Help Others, Please Share

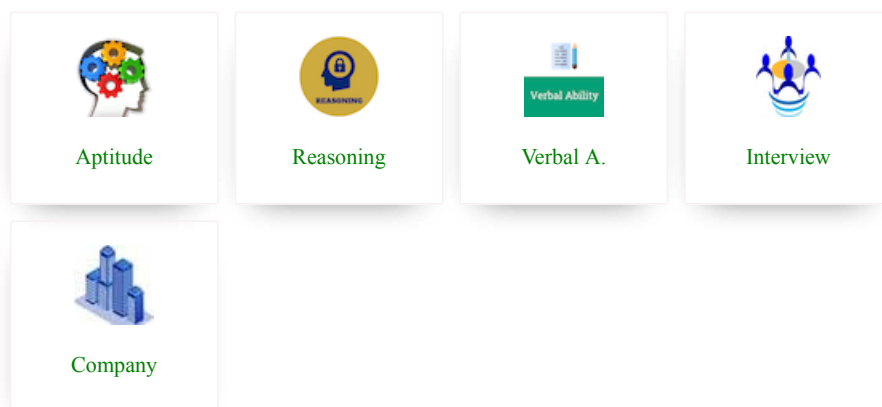




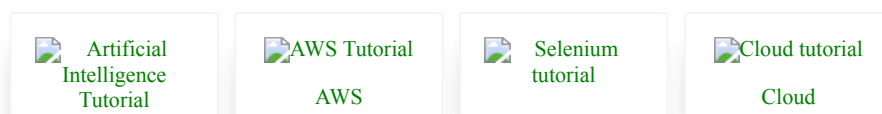
Learn Latest Tutorials











Preparation

























Trending Technologies



AI		Selenium	
 Hadoop tutorial Hadoop	 ReactJS Tutorial ReactJS	 Data Science Tutorial D. Science	 Angular 7 Tutorial Angular 7
 Blockchain Tutorial Blockchain	 Git Tutorial Git	 Machine Learning Tutorial ML	 DevOps Tutorial DevOps

B.Tech / MCA

 DBMS tutorial DBMS	 Data Structures tutorial DS	 DAA tutorial DAA	 Operating System tutorial OS
 Computer Network tutorial C. Network	 Compiler Design tutorial Compiler D.	 Computer Organization and Architecture COA	 Discrete Mathematics Tutorial D. Math.
 Ethical Hacking Tutorial E. Hacking	 Computer Graphics Tutorial C. Graphics	 Software Engineering Tutorial Software E.	 html tutorial Web Tech.
 Cyber Security tutorial Cyber Sec.	 Automata Tutorial Automata	 C Language tutorial C	 C++ tutorial C++
 Java tutorial Java	 .Net Framework tutorial .Net	 Python tutorial Python	 List of Programs Programs
 Control Systems tutorial Control S.	 Data Mining Tutorial Data Mining		

JavaScript Functions

[< Previous](#)[Next >](#)

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 myFunction(p1, p2) {  
  return p1 * p2;    // The function returns the product of p1 and 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.

A Function is much the same as a Procedure or a Subroutine, in other programming languages.

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:

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

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

The result in x will be:

12

[Try it Yourself »](#)

Why Functions?

You can reuse code: Define the code once, and use it many times.

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

Example

Convert Fahrenheit to Celsius:


```
function toCelsius(fahrenheit) {  
    return (5/9) * (fahrenheit-32);  
}  
document.getElementById("demo").innerHTML = toCelsius(77);
```

[Try it Yourself »](#)

The () Operator Invokes the Function

Using the example above, `toCelsius` refers to the function object, and `toCelsius()` refers to the function result.

Accessing a function without () will return the function object instead of the function result.

Example

```
function toCelsius(fahrenheit) {  
    return (5/9) * (fahrenheit-32);  
}  
document.getElementById("demo").innerHTML = toCelsius;
```

[Try it Yourself »](#)

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:

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

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

```
var 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() {  
  var 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.

Test Yourself With Exercises

Exercise:

Execute the function named `myFunction` .

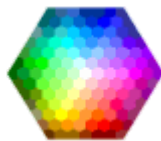
```
function myFunction() {  
  alert("Hello World!");  
}  
;
```

[Submit Answer »](#)

[Start the Exercise](#)

[< Previous](#)[Next >](#)

COLOR PICKER



LIKE US





HOW TO

Tabs
Dropdowns
Accordions
Side Navigation
Top Navigation
Modal Boxes
Progress Bars
Parallax
Login Form
HTML Includes
Google Maps
Range Sliders
Tooltips
Slideshow
Filter List
Sort List

Certificates

HTML
CSS
JavaScript
Python
SQL

PHP
And more

[REPORT ERROR](#)

[FORUM](#)

[ABOUT](#)

[SHOP](#)

Top Tutorials

[HTML Tutorial](#)
[CSS Tutorial](#)
[JavaScript Tutorial](#)
[How To Tutorial](#)
[SQL Tutorial](#)
[Python Tutorial](#)
[W3.CSS Tutorial](#)
[Bootstrap Tutorial](#)
[PHP Tutorial](#)
[Java Tutorial](#)
[C++ Tutorial](#)
[jQuery Tutorial](#)

Top References

[HTML Reference](#)
[CSS Reference](#)
[JavaScript Reference](#)
[SQL Reference](#)
[Python Reference](#)
[W3.CSS Reference](#)
[Bootstrap Reference](#)
[PHP Reference](#)
[HTML Colors](#)

[Java Reference](#)
[Angular Reference](#)
[jQuery Reference](#)

Top Examples

[HTML Examples](#)
[CSS Examples](#)
[JavaScript Examples](#)
[How To Examples](#)
[SQL Examples](#)
[Python Examples](#)
[W3.CSS Examples](#)
[Bootstrap Examples](#)
[PHP Examples](#)
[Java Examples](#)
[XML Examples](#)
[jQuery Examples](#)

Web Certificates

[HTML Certificate](#)
[CSS Certificate](#)
[JavaScript Certificate](#)
[SQL Certificate](#)
[Python Certificate](#)
[PHP Certificate](#)
[Bootstrap Certificate](#)
[XML Certificate](#)
[jQuery Certificate](#)

[Get Certified »](#)

W3Schools is optimized for learning and training. Examples might be simplified to improve reading and learning. Tutorials, references, and examples are constantly reviewed to avoid errors, but we cannot warrant full correctness of all content. While using W3Schools, you agree to have read and accepted our terms of use, cookie and privacy policy.

Copyright 1999-2020 by Refsnes Data. All Rights Reserved.
W3Schools is Powered by W3.CSS.

