**UNIT 1: JAVASCRIPT**

1. Introduction
   1. History of javascript
   2. Features of javascript
2. Programming vs scripting
3. Basic programming constructs in javaScript
   1. Displaying output in javascript
   2. Basics
      1. Statement
      2. Case sensitivity
      3. Semicolon
      4. Comments
      5. Variables
      6. Strings
      7. keywords
   3. Conditional Logic
   4. Arrays
   5. Loops
      1. For loop
      2. For … in loop
      3. While loop
      4. Do while loop
   6. Functions
4. Objects
   1. Creation
   2. Properties
5. Tokens
6. Summary
7. Glossary (karna h)
8. Check your progress
9. References/bibliography
10. Suggested readings
11. Model questions

After reading this unit you will be able to:

* Recognize and remember syntax of javascript.
* Include javascript in your HTML page.
* Create your own interactive web page.
* Develop logical thinking while using logical constructs of javascript.

1. ***INTRODUCTION:***

JavaScript is the programming language that enables web pages to Interact with user. It makes your page more interactive. It was created in 1995 for web based programming and today it also runs on your phone applications and robots.

***Features:***

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

JavaScript can be implemented using JavaScript statements that are placed within the <script>... </script> HTML tags in a web page. You can place the <script> tags anywhere in your web page, but it is a good practice to keep it within <head> tags. The <script> tag alerts the browser program to start interpreting all the text between these tags as a script.

The syntax of our JavaScript should be -

*<script> JavaScript code </script>*

The script tag contains two attributes:

* Language: This attribute specifies the scripting language we are using. Here it is javascript.
* Type: This attribute value should be set to "text/javascript".

So now our JavaScript syntax is:

*<script language="javascript" type="text/javascript">*

*JavaScript code*

*</script>*

**Example:**

Let us take a sample example to print "Hello World".

we call a function document.write which writes a string into our HTML document.

<html>

<body>

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

document.write ("Hello World!")

</script>

</body>

</html>

This code will produce the following result:

Hello World!

1. ***PROGRAMMING VS SCRIPTING***
2. **BASIC PROGRAMMING CONSTRUCTS IN JAVASCRIPT**

## JavaScript Can Change HTML Content.

## How to display your output in JavaScript

JavaScript can "display" data in different ways:

1. Writing into an HTML element, using **innerHTML**.
2. Writing into the HTML output using **document.write()**.
3. Writing into an alert box, using **window.alert()**.
4. **Using inner.HTML**

To access an HTML element, JavaScript can use the d**ocument.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>

## 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 fully loaded, will **delete all existing HTML**:

## 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>

**Example** : JavaScript program to calculate multiplication and division of two numbers (input from user).

<!DOCTYPE html>

<html>

<head>

<meta charset=utf-8 />

<title>JavaScript program to calculate multiplication and division of two numbers </title>

<style type="text/css">

body {margin: 30px;}

</style>

<script type = “text/javascript” language=”javascript”>

function multiplyBy()

{

num1 = document.getElementById("firstNumber").value;

num2 = document.getElementById("secondNumber").value;

document.getElementById("result").innerHTML = num1 \* num2;

}

function divideBy()

{

num1 = document.getElementById("firstNumber").value;

num2 = document.getElementById("secondNumber").value;

document.getElementById("result").innerHTML = num1 / num2;

}

</script>

</head>

<body>

<form>

1st Number : <input type="text" id="firstNumber" /><br>

2nd Number: <input type="text" id="secondNumber" /><br>

<input type="button" onClick="multiplyBy()" Value="Multiply" />

<input type="button" onClick="divideBy()" Value="Divide" />

</form>

<p>The Result is : <br>

<span id = "result"></span>

</p>

</body>

</html>

* 1. **Basics: Before starting with java script what we should know is:**
     1. **Statement**

A statement is a *line of code*.

Example:

var hello = "Hello";

var world = "World";

// Message equals "Hello World"

var message = hello + " " + world;

This code can be executed by another program called an *interpreter* that will read the code, and execute all the statements in the right order.

## *JavaScript is Case Sensitive*

All JavaScript identifiers are **case sensitive**.

The variables **lastName** and **lastname**, are two different variables.

var lastname, lastName;  
lastName = "Doe";  
lastname = "Peterson";

JavaScript does not interpret **VAR** or **Var** as the keyword **var**.

## Semicolons

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

var a, b, c;  
a = 5;  
b = 6;  
c = a + b;

* + 1. **Comments**

Comments are statements that will not be executed by the interpreter, comments are used to mark annotations for other programmers or small descriptions of what your code does, thus making it easier for others to understand what your code does.

In Javascript, comments can be written in 2 different ways:

1. Line starting with //:

// This is a comment, it will be ignored by the interpreter

1. Section of code starting with /\*and ending with \*/, this method is used for multi-line comments:

/\*

This is a multi-line comment,

it will be ignored by the interpreter

\*/

* + 1. **Variables**

Variables are containers for changeable values. Variables can hold all kind of values and also the results of computations. Variables have a name and a value separated by an equal sign (=).

Use the **var** keyword only for declaration or initialization, first time you use the variable in a document. You should not re-declare same variable twice.

JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

Example: var value;

var result = 2;

You can also declare multiple variables with the same var keyword as follows −

var value, result;

*Example*: this example defines two variables, after computing their addition the result is stored in third variable.

var x = 5;

var y = 6;

var result = x + y;

**Types of Variables**

Computers are sophisticated and can make use of more complex variables than just numbers. This is where variable types come in. Variables come in several types and different languages support different types.

The most common types are:

1. **Numbers**
   * 1. **Float**: a number, like 1.21323, 4, -33.5, 100004 or 0.123
     2. **Integer**: a number like 1, 12, -33, 140 but not 1.233
2. **String**: a line of text like "boat", "elephant" or "damn, you are tall!”. (We will discuss more on strings in later section).
3. **Boolean**: either true or false, but nothing else
4. **Arrays**: a collection of values like: 1,2,3,4,'I am bored now'
5. **Objects**: a representation of a more complex object
6. **null**: a variable that contains null contains no valid Number, String, Boolean, Array, or Object
7. **undefined**: the undefined value is obtained when you use an object property that does not exist, or a variable that has been declared, but has no value assigned to it.

Exercise: create a variable named ‘myvar’ using keyword var.

* + 1. **Strings**

This is a string.

“I love programming”;

You can define strings using var keyword either using single or double quotes.

// Single quotes

var str = 'Please Enter your name;

// Double quotes

var print = "Thank you for your input";

**Concatenation**: Concatenation is adding two or more strings together to creating a larger string. This is done using the + operator.

var message = ' wow ' + 'JS is easy to learn' + 'and ' + 'easy to use;

example:

var firstName = "Alok";

var lastName = "Kumar";

var fullName = firstName + " " + lastName;

You can check the length of your string in Javascript (length is the number of characters in string)

Just use the property .length

var sizeofString = 'I Love India'.length;

* + 1. **Keywords**

There are some reserved words in every language which cannot be used as variables. Javascript keywords are:

|  |  |  |  |
| --- | --- | --- | --- |
| Abstract | arguments | boolean | Break |
| Byte | Case | catch | Char |
| Const | Continue | debugger | Default |
| Delete | Do | double | Else |
| Eval | False | Final | Finally |
| Float | For | function | Goto |
| If | implements | In | instanceof |
| Int | Interface | Long | Native |
| New | Null | package | Private |
| Protected | Public | return | short |
| Static | Switch | synchronized | this |
| Throw | Throws | transient | true |
| Try | Typeof | Var | void |
| Volatile | While | With | yield |

* 1. **Conditional logic**

Programming in any language needs to implement logic in several ways. Conditions are checks to ensure correct flow of functioning.

There are many ways to check conditions:

* + 1. **If**
    2. **Else**
    3. **Logical comparison operator**
  1. **Arrays**

Loops are used for statements that need to be repeated. one or more variable in the loop changes so that you can check condition to stop. You can run the same code again and again, each time with a different value. There are three types of loops.

* 1. **Loops** 
     1. **For loop**

The easiest form of a loop is the for statement. This one has a syntax that is similar to an if statement, but with more options:

syntax:

for (*statement 1*;*statement 2*;*statement 3*) {  
    *code block to be executed*  
}

**Statement 1** is executed before the loop starts.

**Statement 2** defines the end condition when the loop had to stop.

**Statement 3** is executed each time after the loop has been executed.

Example: for loop to print numbers from 1 to 5

<!DOCTYPE html>

<html>

<body>

<p id="output"></p>

<script>

var text = "";

var i;

for (i = 0; i < 5; i++) {

text += "The number is " + i + "<br>";

}

document.getElementById("output").innerHTML = text;

</script>

</body>

</html>

* + 1. **For .. in**

The for...in loop is used to loop through an object's properties. As we have not discussed Objects yet, you may not feel comfortable with this loop. But once you understand how objects behave in JavaScript, you will find this loop very useful.

Syntax:

The syntax of ‘for..in’ loop is:

for (variablename in object)

{ statement or block to execute }

* + 1. **While loop**

While Loops repetitively execute a block of code as long as a specified condition is true.

while(condition){

// do it as long as condition is true

}

For example, the loop in this example will repetitively execute its block of code as long as the variable i is less than 5:

var i = 0, x = "";

while (i < 5) {

x = x + "The number is " + i;

i++;

}

* + 1. **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. It then repeats the loop as long as the condition is true:

do {

// code block to be executed

} while (condition);

Note: Be careful to avoid infinite looping if the condition is always true!

Example: print numbers less than 10 using do...while loop:

var i = 0;

do {

document.write(i + " ");

i++; // incrementing i by 1

} while (i < 10);

Note: i = i + 1 can be written i++.

* 1. **Functions**

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions. Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like alert() and write() in the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once.

JavaScript allows us to write our own functions as well. This section explains how to write your own functions in JavaScript.

* + 1. **Function definition**

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

**Syntax:**

<script type="text/javascript">

function functionname(parameter-list)

{

statements

}

</script>

Example:

Try the following example. It defines a function called sayHello that takes no parameters:

<script type="text/javascript">

function sayHello()

{

alert("Hello there");

}

</script>

* + 1. **Function calling**

Calling a Function To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

<html>

<head>

<script type="text/javascript">

function sayHello()

{

document.write ("Hello there!");

}

</script>

</head>

<body>

<p>Click the following button to call the function</p>

<form>

<input type="button" onclick="sayHello()" value="Say Hello">

</form>

</body>

</html>

* + 1. **Function parameters**

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma. Example Try the following example. We have modified our sayHello function here. Now it takes two parameters.

<html>

<head>

<script type="text/javascript">

function sayHello(name, age)

{

document.write (name + " is " + age + " years old.");

}

</script>

</head>

<body>

<p>Click the following button to call the function</p>

<form>

<input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">

</form>

</body>

</html>

* + 1. **cdsfdsg**
  1. **sdsdgf**

1. **Objects**
2. **Tokens**

# Arrays

## 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.

## 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"];

An **index** refers to a spot in the array. indices logically progress one by one, but it should be noted that the first index in an array is 0, as it is in most languages. Brackets [] are used to signify you are referring to an index of an array.

// This is an array of strings

var fruits = ["apple", "banana", "pineapple", "strawberry"];

// We set the variable banana to the value of the second element of

// the fruits array. Remember that indices start at 0, so 1 is the

// second element. Result: banana = "banana"

var banana = fruits[1];

Arrays have a property called length, and it's pretty much exactly as it sounds, it's the length of the array.

var array = [1 , 2, 3];

// Result: l = 3

var l = array.length;

1. **Objects**

In real life, a car is an **object**.

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

|  |  |  |
| --- | --- | --- |
| **Object** | **Properties** | **Methods** |
| https://www.w3schools.com/js/objectExplained.gif | car.name = Fiat  car.model = 500  car.weight = 850kg  car.color = white | car.start()  car.drive()  car.brake()  car.stop() |

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.

**Example ::**

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

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

var car = "Fiat";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_variable)

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

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

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_object)

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

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

**We will discuss objects in detail under unit 2.**

1. **Tokens**

Tokens are the smallest individual words, phrases, or characters that JavaScript can understand. When JavaScript is interpreted, the browser parses the script into these tokens while ignoring comments and white space. JavaScript tokens fit in five categories:

* 1. **Identifiers**

Identifiers are simply names that represent variables, methods, or objects. They consist of a combination of characters and digits. Some names are already built into the JavaScript language and are therefore reserved these identifiers are called as "Keywords". Aside from these keywords, you can define your own creative and meaningful identifiers. Of course, you have a couple of rules to follow. You must begin all identifiers with either a letter or underscore C).

You can then use letters, digits, or underscores for all subsequent characters. Letters include all uppercase characters, A through Z, and all lowercase characters, a through z. Digits include the characters 0 through 9. The table below shows some examples of valid and invalid identifiers.

|  |  |  |
| --- | --- | --- |
| Valid | Invalid | Reason |
| current- WebSite | current WebSite | contains a space |
| NumberOfHits | #ofIslands | pound sign is prefixed |
| N | 2bOrNotToBe | begins with a number |
| N | Return | Key Word |

* 1. Keywords

Keywords are predefined identifiers that make up the core of a programming language. In JavaScript, they perform unique functions such as declaring new variables and functions, making decisions based on the present state of the computer, or starting a repetitive loop inside your application. Keywords, which are built into JavaScript, are always available for use by the programmer but must follow the correct syntax.

|  |  |  |
| --- | --- | --- |
| Break  continue  else  false  for  function | If  in  int  new  null  return | this  true  var  while  with |

* 1. **Literals**

Literals are data comprised of numbers or strings used to represent fixed values in JavaScript. They are values that do not change during the execution of your scripts. The following five sections contain descriptions and examples of the different types of literals that you can use

**Integer Literals**

Integers can be expressed in either decimal (base 10), octal (base 8), or hexadecimal (base 16) format. An integer literal in decimal format can include any sequence of digits that does not begin with a 0 (zero). A zero in front of an integer literal designates octal form. The integer itself can include a sequence of the digits 0 through 7. To designate hexadecimal, Ox (or OX) is used before the integer. Hexadecimal integers can include digits 0 through 9 along with the letters. a through f or A through F. Some examples include

|  |  |
| --- | --- |
| Decimal (base 10) | 33,2139 |
| Octal (base 8) | 071,03664 |
| Hexadecimal (base 16) | Ox7b8, OX395 |

**Floating-Point Literals**

Floating-point literals represent decimal numbers with fractional parts. They can be expressed in either standard or scientific notation. With scientific notation, use either e or E to designate the exponent. Both the decimal number and exponent can be either signed or unsigned as shown in the examples:

3405.673

-1.958

8.3200e+ 11

8.3200e11

9.98E-12

**Boolean Literals**

JavaScript implements Boolean data types and therefore supports the two literals, trueand false.

They represent the Boolean values 1 and 0, respectively. The true and false keywords must appear in lowercase. As a result, the capitalized words TRUE and FALSE are left open to define as your own identifiers, but it is not recommended.

**String Literals**

A string literal is zero or more characters enclosed in double (") or single (') quotes. JavaScript gives you this option, but you must use the same type of quote to surround each string. The following are examples of string literals enclosed in quotes:

"virtual communities"

'virtual communities'

"Look, up in the sky!"

**Special Characters**

When writing scripts, you might sometimes need to tell the computer to use a special character or keystroke such as a tab or carriage return. To do this, use a backslash in front of one of the special characters as shown in the following list:

|  |  |
| --- | --- |
| \b | indicates a backspace |
| \f | indicates a form feed |
| \n | indicates a new-line character |
| \r | indicates a carriage return |
| \t | indicates a tab character |
|  |  |
|  |  |

* 1. **Operators**
  2. Separators
  3. **Identifiers**

**Literals**

***Points to remember***

* JavaScript is a client-side as well as server side scripting language.
* JavaScript is also an Object based Programming language.
* JavaScript is a *“loosely typed”* language, which means that you don't have to explicitly declare what type of data the variables are. You just need to use the var keyword to indicate that you are declaring a variable, and the interpreter will work out with its associated properties.
* You can define strings in JavaScript by enclosing the text in single quotes or double quotes. Ex: var name=’Ankit’; or var name=”Ankit”;

***Test yourself***

1. Is JavaScript a case-sensitive language?

A – True B- False

1. Which built-in method returns the length of the string?

[**A** - length()](javascript:void(0);) [**B** - size()](javascript:void(0);) [**C** - index()](javascript:void(0);) [**D** - None of the above.](javascript:void(0);)

1. Sdgsdg
2. sfadsd

**UNIT 2: JAVASCRIPT**

1. Introduction
2. Javascript events
3. Objects
4. Client validations
5. Cookies
6. DHTML
7. Summary
8. Glossary (karna h)
9. Check your progress
10. References/bibliography
11. Suggested readings
12. Model questions

After reading this unit you will be able to:

* Recognize and remember syntax of javascript.
* Include javascript in your HTML page.
* Create your own interactive web page.
* Develop logical thinking while using logical constructs of javascript.

When a user visit your website, they do things like click on text and images and given links, hover over things etc. These are examples of what JavaScript calls events.

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.

## onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

### Example

Try the following example.

<html>

<head>

<script type="text/javascript">

<!--

function sayHello() {

alert("Hello World")

}

//-->

</script>

</head>

<body>

<p>Click the following button and see result</p>

<form>

<input type="button" onclick="sayHello()" value="Say Hello" />

</form>

</body>

</html>

## onsubmit Event type

**onsubmit** is an event that occurs when you try to submit a form. You can put your form validation against this event type.

### Example

The following example shows how to use onsubmit. Here we are calling a **validate()** function before submitting a form data to the webserver. If **validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

Try the following example.

<html>

<head>

<script type="text/javascript">

<!--

function validation() {

all validation goes here

.........

return either true or false

}

//-->

</script>

</head>

<body>

<form method="POST" action="t.cgi" onsubmit="return validate()">

.......

<input type="submit" value="Submit" />

</form>

</body>

</html>

## onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the**onmouseout** triggers when you move your mouse out from that element. Try the following example.

<html>

<head>

<script type="text/javascript">

<!--

function over() {

document.write ("Mouse Over");

}

function out() {

document.write ("Mouse Out");

}

//-->

</script>

</head>

<body>

<p>Bring your mouse inside the division to see the result:</p>

<div onmouseover="over()" onmouseout="out()">

<h2> This is inside the division </h2>

</div>

</body>

</html>

## HTML 5 Standard Events

The standard HTML 5 events are listed here for your reference. Here script indicates a Javascript function to be executed against that event.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Value** | **Description** |
| Offline | script | Triggers when the document goes offline |
| Onabort | script | Triggers on an abort event |
| onafterprint | script | Triggers after the document is printed |
| onbeforeonload | script | Triggers before the document loads |
| onbeforeprint | script | Triggers before the document is printed |
| onblur | script | Triggers when the window loses focus |
| oncanplay | script | Triggers when media can start play, but might has to stop for buffering |
| oncanplaythrough | script | Triggers when media can be played to the end, without stopping for buffering |
| onchange | script | Triggers when an element changes |
| onclick | script | Triggers on a mouse click |
| oncontextmenu | script | Triggers when a context menu is triggered |
| ondblclick | script | Triggers on a mouse double-click |
| ondrag | script | Triggers when an element is dragged |
| ondragend | script | Triggers at the end of a drag operation |
| ondragenter | script | Triggers when an element has been dragged to a valid drop target |
| ondragleave | script | Triggers when an element is being dragged over a valid drop target |
| ondragover | script | Triggers at the start of a drag operation |
| ondragstart | script | Triggers at the start of a drag operation |
| ondrop | script | Triggers when dragged element is being dropped |
| ondurationchange | script | Triggers when the length of the media is changed |
| onemptied | script | Triggers when a media resource element suddenly becomes empty. |
| onended | script | Triggers when media has reach the end |
| onerror | script | Triggers when an error occur |
| onfocus | script | Triggers when the window gets focus |
| onformchange | script | Triggers when a form changes |
| onforminput | script | Triggers when a form gets user input |
| onhaschange | script | Triggers when the document has change |
| oninput | script | Triggers when an element gets user input |
| oninvalid | script | Triggers when an element is invalid |
| onkeydown | script | Triggers when a key is pressed |
| onkeypress | script | Triggers when a key is pressed and released |
| onkeyup | script | Triggers when a key is released |
| onload | script | Triggers when the document loads |
| onloadeddata | script | Triggers when media data is loaded |
| onloadedmetadata | script | Triggers when the duration and other media data of a media element is loaded |
| onloadstart | script | Triggers when the browser starts to load the media data |
| onmessage | script | Triggers when the message is triggered |
| onmousedown | script | Triggers when a mouse button is pressed |
| onmousemove | script | Triggers when the mouse pointer moves |
| onmouseout | script | Triggers when the mouse pointer moves out of an element |
| onmouseover | script | Triggers when the mouse pointer moves over an element |
| onmouseup | script | Triggers when a mouse button is released |
| onmousewheel | script | Triggers when the mouse wheel is being rotated |
| onoffline | script | Triggers when the document goes offline |
| onoine | script | Triggers when the document comes online |
| ononline | script | Triggers when the document comes online |
| onpagehide | script | Triggers when the window is hidden |
| onpageshow | script | Triggers when the window becomes visible |
| onpause | script | Triggers when media data is paused |
| onplay | script | Triggers when media data is going to start playing |
| onplaying | script | Triggers when media data has start playing |
| onpopstate | script | Triggers when the window's history changes |
| onprogress | script | Triggers when the browser is fetching the media data |
| onratechange | script | Triggers when the media data's playing rate has changed |
| onreadystatechange | script | Triggers when the ready-state changes |
| onredo | script | Triggers when the document performs a redo |
| onresize | script | Triggers when the window is resized |
| onscroll | script | Triggers when an element's scrollbar is being scrolled |
| onseeked | script | Triggers when a media element's seeking attribute is no longer true, and the seeking has ended |
| onseeking | script | Triggers when a media element's seeking attribute is true, and the seeking has begun |
| onselect | script | Triggers when an element is selected |
| onstalled | script | Triggers when there is an error in fetching media data |
| onstorage | script | Triggers when a document loads |
| onsubmit | script | Triggers when a form is submitted |
| onsuspend | script | Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched |
| ontimeupdate | script | Triggers when media changes its playing position |
| objectsonundo | script | Triggers when a document performs an undo |
| onunload | script | Triggers when the user leaves the document |
| onvolumechange | script | Triggers when media changes the volume, also when volume is set to "mute" |
| onwaiting | script | Triggers when media has stopped playing, but is expected to resume |

**Objects ::**

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

All JavaScript values, except primitives, are objects.

**Objects are variable that contain multiple values.**

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

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

JavaScript objects are containers for named values, called properties and methods.

**Object Properties**

**The named values, in JavaScript objects, are called properties.**

|  |  |
| --- | --- |
| **Property** | **Value** |
| **firstName** | **John** |
| **lastName** | **Doe** |
| **age** | **50** |
| **eyeColor** | **Blue** |

# *Creation*

There are two ways to create an object in JavaScript:

1. literal
2. var object = {};
3. // Yes, simply a pair of curly braces!

**Note:** this is the **recomended** way.

1. and object-oriented
2. var object = new Object();

**Note:** it's almost like Java.

# Creation

There are two ways to create an object in JavaScript:

1. literal
2. var object = {};
3. // Yes, simply a pair of curly braces!

**Note:** this is the **recomended** way.

1. and object-oriented
2. var object = new Object();

**Note:** it's almost like Java.

# Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is −

objectName.objectProperty = propertyValue;

## Accessing Object Properties

You can access object properties in two ways:

*objectName.propertyName*

or

*objectName["propertyName"]*

### Example1

person.lastName;

Object's property is a propertyName: propertyValue pair, where **property name can be only a string**. If it's not a string, it gets casted into a string. You can specify properties **when creating** an object **or later**. There may be zero or more properties separated by commas.

var language = {

name: 'JavaScript',

isSupportedByBrowsers: true,

createdIn: 1995,

author:{

firstName: 'Brendan',

lastName: 'Eich'

},

// Yes, objects can be nested!

getAuthorFullName: function(){

return this.author.firstName + " " + this.author.lastName;

}

// Yes, functions can be values too!

};

## Object Methods

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the **this** keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

**For example** − Following is a simple example to show how to use the **write()** method of document object to write any content on the document.

document.write("This is test");

## User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called **Object**.

### The new Operator

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

var employee = new Object();

var books = new Array("C++", "Perl", "Java");

var day = new Date("August 15, 1947");

### The Object() Constructor

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called **Object()**to build the object. The return value of the **Object()** constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the **var** keyword.

### Example 1

Try the following example; it demonstrates how to create an Object.

<html>

<head>

<title>User-defined objects</title>

<script type="text/javascript">

var book = new Object(); // Create the object

book.subject = "Perl"; // Assign properties to the object

book.author = "Mohtashim";

</script>

</head>

<body>

<script type="text/javascript">

document.write("Book name is : " + book.subject + "<br>");

document.write("Book author is : " + book.author + "<br>");

</script>

</body>

</html>

### Output

Book name is : Perl

Book author is : Mohtashim

### Example 2

This example demonstrates how to create an object with a User-Defined Function. Here **this** keyword is used to refer to the object that has been passed to a function.

<html>

<head>

<title>User-defined objects</title>

<script type="text/javascript">

function book(title, author){

this.title = title;

this.author = author;

}

</script>

</head>

<body>

<script type="text/javascript">

var myBook = new book("Perl", "Mohtashim");

document.write("Book title is : " + myBook.title + "<br>");

document.write("Book author is : " + myBook.author + "<br>");

</script>

</body>

</html>

### Output

Book title is : Perl

Book author is : Mohtashim

## Defining Methods for an Object

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complete the definition of an object by assigning methods to it.

### Example

Try the following example; it shows how to add a function along with an object.

<html>

<head>

<title>User-defined objects</title>

<script type="text/javascript">

// Define a function which will work as a method

function addPrice(amount){

this.price = amount;

}

function book(title, author){

this.title = title;

this.author = author;

this.addPrice = addPrice; // Assign that method as property.

}

</script>

</head>

<body>

<script type="text/javascript">

var myBook = new book("Perl", "Mohtashim");

myBook.addPrice(100);

document.write("Book title is : " + myBook.title + "<br>");

document.write("Book author is : " + myBook.author + "<br>");

document.write("Book price is : " + myBook.price + "<br>");

</script>

</body>

</html>

### Output

Book title is : Perl

Book author is : Mohtashim

Book price is : 100

***Example: we will take an example of string object in java to understand the functionality of objects.***

***String is an array of characters. (tutorialspoint javascr pdf page 136)***

**Client validations**

**Validating data is a process to ensure security and enhanced user experience by providing services accurately and flawlessly. Data validation can be done on either client end or at server end.**

**In HTML you created forms to receive data from user and that data is send to server for further processing. If the data is validated at server end then in case of incorrect entries,** the server will send all the data back to the client and the form has to be refilled and resubmitted by user with correct information. This is a lengthy process by which a lot of burden is put on the server.

**Javasc provides a way to validate data at user end for fast processing and good user experience. Data validation is done with the help of events that are provided in javascript.**

**Validation is done to ensure that the data user had filled is correct, relevant and according to requirement. Validation can help us in**

* **Check if the field is left empty.**
* **Check if the correct value is entered in field. Ex email is entered with right credentials.**
* **Check if the user has entered correct range of value (ex age can range from 1 to 120).**
* **Check if the user has entered text in numeric field.**

**Let us take an example of an html form.**

**(code to create a simple form with two input fields a combo and a submit button).**

**Output printscreen here.**

**Events wale portion mai saare events batane hain jo yahan use honge with reference use them.**

**Javascript code for validation here…..**

**We can validate form elements with the helpof [predefined constraints.**

## Example

## The required Attribute

The **required** attribute specifies that an input field must be filled out before submitting the form.

The required attribute works with the following input types: text, search, url, tel, email, password, date pickers, number, checkbox, radio, and file.

OperaSafariChromeFirefox

### Example

A required input field:

Username: <input type="text" name="usrname" required>

## The maxlength Attribute

The **maxlength** attribute specifies the maximum allowed length for the input field:

### Example

<form action="">  
First name:<br>  
<input type="text" name="firstname" maxlength="10">  
</form>

**Cookies**

**What are Cookies?**

**Web Browsers and Servers use HTTP protocol to communicate and HTTP is a**

**stateless protocol. But for a commercial website, it is required to maintain**

**session information among different pages. For example, one user registration**

**ends after completing many pages. But how to maintain users' session**

**information across all the web pages.**

**In many situations, using cookies is the most efficient method of remembering**

**and tracking preferences, purchases, commissions, and other information**

**required for better visitor experience or site statistics.**

**How It Works?**

**Your server sends some data to the visitor's browser in the form of a cookie. The**

**browser may accept the cookie. If it does, it is stored as a plain text record on**

**the visitor's hard drive. Now, when the visitor arrives at another page on your**

**site, the browser sends the same cookie to the server for retrieval. Once**

**retrieved, your server knows/remembers what was stored earlier.**

**Cookies are a plain text data record of 5 variable-length fields:**

** Expires: The date the cookie will expire. If this is blank, the cookie will**

**expire when the visitor quits the browser.**

** Domain: The domain name of your site.**

** Path: The path to the directory or web page that set the cookie. This may**

**be blank if you want to retrieve the cookie from any directory or page.**

** Secure: If this field contains the word "secure", then the cookie may only**

**be retrieved with a secure server. If this field is blank, no such restriction**

**exists.**

** Name=Value: Cookies are set and retrieved in the form of key-value**

**pairs.**

**Cookies were originally designed for CGI programming. The data contained in a**

**cookie is automatically transmitted between the web browser and the web**

**server, so CGI scripts on the server can read and write cookie values that are**

**stored on the client.**

**15. COOKIES**

**Javascript**

**83**

**JavaScript can also manipulate cookies using the cookie property of**

**the Document object. JavaScript can read, create, modify, and delete the**

**cookies that apply to the current web page.**

**Storing Cookies**

**The simplest way to create a cookie is to assign a string value to the**

**document.cookie object, which looks like this.**

**document.cookie = "key1=value1;key2=value2;expires=date";**

**Here the expires attribute is optional. If you provide this attribute with a valid**

**date or time, then the cookie will expire on a given date or time and thereafter,**

**the cookies' value will not be accessible.**

**Note: Cookie values may not include semicolons, commas, or whitespace. For**

**this reason, you may want to use the JavaScript escape() function to encode**

**the value before storing it in the cookie. If you do this, you will also have to use**

**the corresponding unescape() function when you read the cookie value.**

**Example**

**Try the following. It sets a customer name in an input cookie.**

**<html>**

**<head>**

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

**<!--**

**function WriteCookie()**

**{**

**if( document.myform.customer.value == "" ){**

**alert ("Enter some value!");**

**return;**

**}**

**cookievalue= escape(document.myform.customer.value) + ";";**

**document.cookie="name=" + cookievalue;**

**document.write ("Setting Cookies : " + "name=" + cookievalue );**

**}**

**//-->**

**</script>**

**Javascript**

**84**

**</head>**

**<body>**

**<form name="myform" action="">**

**Enter name: <input type="text" name="customer"/>**

**<input type="button" value="Set Cookie" onclick="WriteCookie();"/>**

**</form>**

**</body>**

**</html>**

**Output**

**Enter name:**

**Now your machine has a cookie called name. You can set multiple cookies using**

**multiple key=value pairs separated by comma.**

**Reading Cookies**

**Reading a cookie is just as simple as writing one, because the value of**

**the document.cookie object is the cookie. So you can use this string whenever**

**you want to access the cookie. The document.cookie string will keep a list**

**of name=value pairs separated by semicolons, where name is the name of a**

**cookie and value is its string value.**

**You can use strings' split() function to break a string into key and values as**

**follows:**

**Example**

**Try the following example to get all the cookies.**

**<html>**

**<head>**

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

**<!--**

**function ReadCookie()**

**{**

**var allcookies = document.cookie;**

**document.write ("All Cookies : " + allcookies );**

**Set**

**Cookie**

**Javascript**

**85**

**// Get all the cookies pairs in an array**

**cookiearray = allcookies.split(';');**

**// Now take key value pair out of this array**

**for(var i=0; i<cookiearray.length; i++){**

**name = cookiearray[i].split('=')[0];**

**value = cookiearray[i].split('=')[1];**

**document.write ("Key is : " + name + " and Value is : " + value);**

**}**

**}**

**//-->**

**</script>**

**</head>**

**<body>**

**<form name="myform" action="">**

**<p> click the following button and see the result:</p>**

**<input type="button" value="Get Cookie" onclick="ReadCookie()"/>**

**</form>**

**</body>**

**</html>**

**Note: Here length is a method of Array class which returns the length of an**

**array. We will discuss Arrays in a separate chapter. By that time, please try to**

**digest it.**

**Output**

**click the following button and see the result:**

**Note: There may be some other cookies already set on your machine. The**

**above code will display all the cookies set on your machine.**

**Get Cookie**

**Javascript**

**86**

**Setting Cookies Expiry Date**

**You can extend the life of a cookie beyond the current browser session by**

**setting an expiration date and saving the expiry date within the cookie. This can**

**be done by setting the ‘expires’ attribute to a date and time.**

**Example**

**Try the following example. It illustrates how to extend the expiry date of a**

**cookie by 1 Month.**

**<html>**

**<head>**

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

**<!--**

**function WriteCookie()**

**{**

**var now = new Date();**

**now.setMonth( now.getMonth() + 1 );**

**cookievalue = escape(document.myform.customer.value) + ";"**

**document.cookie="name=" + cookievalue;**

**document.cookie = "expires=" + now.toUTCString() + ";"**

**document.write ("Setting Cookies : " + "name=" + cookievalue );**

**}**

**//-->**

**</script>**

**</head>**

**<body>**

**<form name="formname" action="">**

**Enter name: <input type="text" name="customer"/>**

**<input type="button" value="Set Cookie" onclick="WriteCookie()"/>**

**</form>**

**</body>**

**</html>**

**Javascript**

**87**

**Output**

**Enter Cookie Name:**

**Deleting a Cookie**

**Sometimes you will want to delete a cookie so that subsequent attempts to read**

**the cookie return nothing. To do this, you just need to set the expiry date to a**

**time in the past.**

**Example**

**Try the following example. It illustrates how to delete a cookie by setting its**

**expiry date to one month behind the current date.**

**<html>**

**<head>**

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

**<!--**

**function WriteCookie()**

**{**

**var now = new Date();**

**now.setMonth( now.getMonth() - 1 );**

**cookievalue = escape(document.myform.customer.value) + ";"**

**document.cookie="name=" + cookievalue;**

**document.cookie = "expires=" + now.toUTCString() + ";"**

**document.write("Setting Cookies : " + "name=" + cookievalue );**

**}**

**//-->**

**</script>**

**</head>**

**<body>**

**<form name="formname" action="">**

**Enter name: <input type="text" name="customer"/>**

**<input type="button" value="Set Cookie" onclick="WriteCookie()"/>**

**</form>**

**Set Cookie**

**Javascript**

**88**

**</body>**

**</html>**

**Output**

**Enter Cookie Name:**

**DHTML**

**UNIT 3: JQUERY**

1. Introduction
2. Jquery
   1. What is jquery
   2. Why jquery
   3. Including jquery in a web page
   4. How to write jquery
   5. Selectors
   6. Methods
3. Accessing Jquery libraries
4. Glossary (karna h)
5. Check your progress
6. References/bibliography
7. Suggested readings
8. Model questions

After reading this unit you will be able to:

* Recognize and remember syntax of javascript.
* Include javascript in your HTML page.
* Create your own interactive web page.
* Develop logical thinking while using logical constructs of javascript.

**CHAPTER 1 - Introduction to jquery(basic structure and accessing rule)**

**Introduction**

jQuery is a fast and concise JavaScript Library created by John Resig in 2006 with a nice motto − **Write less, do more**.

jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.

jQuery is a JavaScript toolkit designed to simplify various tasks by writing less code.

jQuery greatly simplifies JavaScript programming.

**As you have learnt HTML, CSS and javascript. You are now able to learn jquery and develop an application based on jquery.**

**We need to understand DOM**

**Explain DOM here with example.**

## The Document Object Model

Definition of DOM as put by the [W3C](http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/introduction.html) is −

The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

The Document Object Model is a tree structure of various elements of HTML as follows −

<html>

<head>

<title>The jQuery Example</title>

</head>

<body>

<div>

<p>This is a paragraph.</p>

<p>This is second paragraph.</p>

<p>This is third paragraph.</p>

</div>

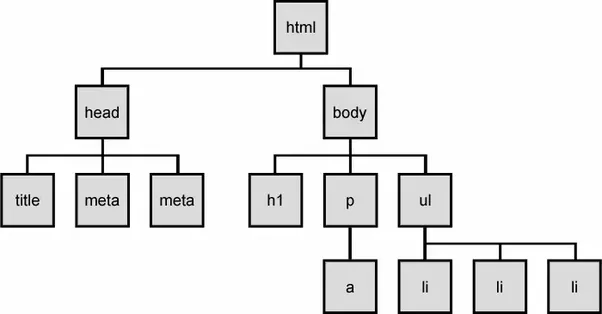
</body>

</html>

This will produce following result −

Following are the important points about the above tree structure −

* The <html> is the ancestor of all the other elements; in other words, all the other elements are descendants of <html>.
* The <head> and <body> elements are not only descendants, but children of <html>, as well.
* Likewise, in addition to being the ancestor of <head> and <body>, <html> is also their parent.
* The <p> elements are children (and descendants) of <div>, descendants of <body> and <html>, and siblings of each other <p> elements.



DOM, Document Object Model comes from XML.

Document Object Model is a in-memory object representation for XML elements loaded into memory. The below example, you have XML, when loaded this into memory, you shall be having Object wrapper for each element. Here person and email are elements, person has two attributes name and age.

DOM creates object for person element with two attributes name and age, person element has 1 child ie email object with text property. A parent element shall have as many child possible, but one child object can have only one parent, hence this forms a Tree hierarchy.

1. **<person** name="steve" age="50"**>**
2. **<email>**steve@example.com**</email>**
3. **</person>**

If you look closely, HTML represent similar form

1. **<html>**
2. **<head>**
3. **<title>**Hello**</title>**
4. **</head>**
5. **<body>**
6. **<div** id="header"**>**
7. **</div>**
8. **</body>**
9. **</html>**

When parsing HTML, browser creates DOM structure, that is nothing but object tree structure, html element is top most parent called root.

You can add/remove HTML elements dynamically into loaded DOM structure to have more interactive web page.

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

* *JavaScript can change all the HTML elements in the page*
* *JavaScript can change all the HTML attributes in the page*
* *JavaScript can change all the CSS styles in the page*
* *JavaScript can remove existing HTML elements and attributes*
* *JavaScript can add new HTML elements and attributes*
* *JavaScript can react to all existing HTML events in the page*
* *JavaScript can create new HTML events in the page*

## What is the HTML DOM?

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

* The HTML elements as **objects**
* The **properties** of all HTML elements
* The **methods** to access all HTML elements
* The **events** for all HTML elements

In other words:**The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

## The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

## Example

The following example changes the content (the innerHTML) of the <p> element with id="demo":

### Example

<html>  
<body>  
  
<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = "Hello World!";  
</script>  
  
</body>  
</html>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_method)

In the example above, getElementById is a **method**, while innerHTML is a **property**.

## The getElementById Method

The most common way to access an HTML element is to use the id of the element.

In the example above the getElementById method used id="demo" to find the element.

## The innerHTML Property

The easiest way to get the content of an element is by using the **innerHTML** property.

The innerHTML property is useful for getting or replacing the content of HTML elements.

The innerHTML property can be used to get or change any HTML element, including <html> and <body>.

## Finding HTML Elements

|  |  |
| --- | --- |
| **Method** | **Description** |
| document.getElementById(id) | Find an element by element id |
| document.getElementsByTagName(name) | Find elements by tag name |
| document.getElementsByClassName(name) | Find elements by class name |

## Changing HTML Elements

|  |  |
| --- | --- |
| **Method** | **Description** |
| element.innerHTML =  new html content | Change the inner HTML of an element |
| element.attribute = new value | Change the attribute value of an HTML element |
| element.setAttribute(attribute, value) | Change the attribute value of an HTML element |
| element.style.property = new style | Change the style of an HTML element |

**EXAMPLE:**

## <!DOCTYPE html>

## <html>

## <body>

## <p id="first">Hello World!</p>

## <p>This example explains getElementById method!</p>

## <p id="new"></p>

## <script>

## var myElement = document.getElementById("first");

## document.getElementById("new").innerHTML = "The text of first paragraph is " + myElement.innerHTML;

## </script>

## </body>

## </html>

Output :

Hello World!

This example explains the **getElementById** method!

The text of first paragraph is Hello World!

## How to use jQuery?

There are two ways to use jQuery.

* **Local Installation** − You can download jQuery library on your local machine and include it in your HTML code.
* **CDN Based Version** − You can include jQuery library into your HTML code directly from Content Delivery Network (CDN).

## Local Installation

* Go to the <https://jquery.com/download/> to download the latest version available.
* Now put downloaded **jquery-2.1.3.min.js** file in a directory of your website, e.g. /jquery.

### Example

Now you can include *jquery* library in your HTML file as follows −

<html>

<head>

<title>The jQuery Example</title>

<script type = "text/javascript" src = "/jquery/jquery-2.1.3.min.js"></script>

<script type = "text/javascript">

$(document).ready(function(){

document.write("Hello, World!");

});

</script>

</head>

<body>

<h1>Hello</h1>

</body>

</html>

This will produce following result −

## CDN Based Version

You can include jQuery library into your HTML code directly from Content Delivery Network (CDN). Google and Microsoft provides content deliver for the latest version.

We are using Google CDN version of the library throughout this tutorial.

### Example

Now let us rewrite above example using jQuery library from Google CDN.

<html>

<head>

<title>The jQuery Example</title>

<script type = "text/javascript" src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>

<script type = "text/javascript">

$(document).ready(function(){

document.write("Hello, World!");

});

</script>

</head>

<body>

<h1>Hello</h1>

</body>

</html>

This will produce following result after loading page –

Hello, World!

***Jquery Selectors***

The jQuery library harnesses the power of Cascading Style Sheets (CSS) selectors to let us quickly and easily access elements or groups of elements in the Document Object Model (DOM).

A jQuery Selector is a function which makes use of expressions to find out matching elements from a DOM based on the given criteria. Simply you can say, selectors are used to select one or more HTML elements using jQuery. Once an element is selected then we can perform various operations on that selected element.

## jQuery Syntax

The jQuery syntax is tailor-made for **selecting** HTML elements and performing some **action** on the element(s).

Basic syntax is: **$(*selector*).*action*()**

* A $ sign to define/access jQuery
* A (*selector*) to "query (or find)" HTML elements
* A jQuery *action*() to be performed on the element(s)

## The Document Ready Event

You might have noticed that all jQuery methods in our examples, are inside a document ready event:

$(document).ready(function(){  
  
   *// jQuery methods go here...*  
  
});

This is to prevent any jQuery code from running before the document is finished loading (is ready).

Execute jQuery only when the complete DOM objects (the complete page has been loaded). You will have to wrap your code in *.ready* function.

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

$(document).ready(function () {

$("#div1").css("border", "2px solid green");

});

</script>

This is the better and safer way to execute jQuery. This makes sure that jQuery code will execute only if complete page has been loaded in the browser so you are rest assured that user will not see any undesired behavior on the page.

## The $() factory function

jQuery selectors start with the dollar sign and parentheses − **$()**. The factory function **$()**makes use of following three building blocks while selecting elements in a given document −

|  |  |
| --- | --- |
| **Sr.No.** | **Selector & Description** |
| 1 | **Tag Name**  Represents a tag name available in the DOM. For example **$('p')** selects all paragraphs <p> in the document. |
| 2 | **Tag ID**  Represents a tag available with the given ID in the DOM. For example **$('#some-id')** selects the single element in the document that has an ID of some-id. |
| 3 | **Tag Class**  Represents a tag available with the given class in the DOM. For example **$('.some-class')** selects all elements in the document that have a class of some-class. |

All the above items can be used either on their own or in combination with other selectors. All the jQuery selectors are based on the same principle except some tweaking.

**NOTE** − The factory function **$()** is a synonym of **jQuery()** function. So in case you are using any other JavaScript library where **$** sign is conflicting with some thing else then you can replace **$** sign by **jQuery** name and you can use function **jQuery()** instead of **$()**.

### Example

Following is a simple example which makes use of Tag Selector. This would select all the elements with a tag name **p** and will set their background to "yellow".

<html>

<head>

<title>The jQuery Example</title>

<script type = "text/javascript"

src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>

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

$(document).ready(function() {

$("p").css("background-color", "yellow");

});

</script>

</head>

<body>

<div>

<p class = "myclass">This is a paragraph.</p>

<p id = "myid">This is second paragraph.</p>

<p>This is third paragraph.</p>

</div>

</body>

</html>

This will produce following result −

## How to use Selectors?

The selectors are very useful and would be required at every step while using jQuery. They get the exact element that you want from your HTML document.

Following table lists down few basic selectors and explains them with examples.

|  |  |
| --- | --- |
| **Sr.No.** | **Selector & Description** |
| 1 | [**Name**](https://www.tutorialspoint.com/jquery/selector-element-name.htm)  Selects all elements which match with the given element **Name**. |
| 2 | [**#ID**](https://www.tutorialspoint.com/jquery/selector-element-id.htm)  Selects a single element which matches with the given **ID**. |
| 3 | [**.Class**](https://www.tutorialspoint.com/jquery/selector-element-class.htm)  Selects all elements which match with the given **Class**. |
| 4 | [**Universal (\*)**](https://www.tutorialspoint.com/jquery/selector-universal.htm)  Selects all elements available in a DOM. |
| 5 | [**Multiple Elements E, F, G**](https://www.tutorialspoint.com/jquery/selector-multiple-elements.htm)  Selects the combined results of all the specified selectors **E, F** or **G**. |

## The element Selector

The jQuery element selector selects elements based on the element name.

You can select all <p> elements on a page like this:

$("p")

**Example**

When a user clicks on a button, all <p> elements will be hidden:

### Example

$(document).ready(function(){  
    $("button").click(function(){  
        $("p").hide();  
    });  
});

[Try it Yourself »](https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_hide_p)

## The #id Selector

The jQuery #id selector uses the id attribute of an HTML tag to find the specific element.

An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element.

To find an element with a specific id, write a hash character, followed by the id of the HTML element:

$("#test")

**Example**

When a user clicks on a button, the element with id="test" will be hidden:

### Example

$(document).ready(function(){  
    $("button").click(function(){  
        $("#test").hide();  
    });  
});

[Try it Yourself »](https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_hide_id)

## The .class Selector

The jQuery class selector finds elements with a specific class.

To find elements with a specific class, write a period character, followed by the name of the class:

$(".test")

**Example**

When a user clicks on a button, the elements with class="test" will be hidden:

### Example

$(document).ready(function(){  
    $("button").click(function(){  
        $(".test").hide();  
    });  
});

[Try it Yourself »](https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_hide_class)

* **$("\*") Selects all elements Try it**
* **$(this) Selects the current HTML element Try it**
* **$("p.intro") Selects all <p> elements with class="intro" Try it**
* **$("p:first") Selects the first <p> element Try it**
* **$("ul li:first") Selects the first <li> element of the first <ul> Try it**
* **$("ul li:first-child") Selects the first <li> element of every <ul> Try it**
* **$(“:button”) selects all <button> elements and <input> elements of type =”button”**

## How to write Functions In a Separate File

If your website contains a lot of pages, and you want your jQuery functions to be easy to maintain, you can put your jQuery functions in a separate .js file.

When we demonstrate jQuery in this tutorial, the functions are added directly into the <head> section. However, sometimes it is preferable to place them in a separate file, like this (use the src attribute to refer to the .js file):

### Example

<head>  
<scriptsrc="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">  
</script>  
<script src="my\_jquery\_functions.js"></script>  
</head>

**CHAPTER 2- Accessing jquery libraries**

## How to call a jQuery library functions?

As almost everything we do when using jQuery reads or manipulates the document object model (DOM), we need to make sure that we start adding events etc. as soon as the DOM is ready.

If you want an event to work on your page, you should call it inside the $(document).ready() function. Everything inside it will load as soon as the DOM is loaded and before the page contents are loaded.

To do this, we register a ready event for the document as follows −

$(document).ready(function() {

// do stuff when DOM is ready

});

To call upon any jQuery library function, use HTML script tags as shown below −

<html>

<head>

<title>The jQuery Example</title>

<script type = "text/javascript"

src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>

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

$(document).ready(function() {

$("div").click(function() {alert("Hello, world!");});

});

</script>

</head>

<body>

<div id = "mydiv">

Click on this to see a dialogue box.

</div>

</body>

</html>

This will produce following result −

## How to use Custom Scripts?

It is better to write our custom code in the custom JavaScript file : **custom.js**, as follows −

/\* Filename: custom.js \*/

$(document).ready(function() {

$("div").click(function() {

alert("Hello, world!");

});

});

Now we can include **custom.js** file in our HTML file as follows −

<html>

<head>

<title>The jQuery Example</title>

<script type = "text/javascript" src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>

<script type = "text/javascript" src = "/jquery/custom.js"></script>

</head>

<body>

<div id = "mydiv">

Click on this to see a dialogue box.

</div>

</body>

</html>

This will produce following result −

**We can use multiple libraries here in a single document.**

***Remember – Conflict possibility.***

Many JavaScript libraries use $ as a function or variable name, just as jQuery does. In jQuery's case, $ is just an alias for jQuery, so all functionality is available without using $.

Run **$.noConflict()** method to give control of the $ variable back to whichever library first implemented it. This helps to make sure that jQuery doesn't conflict with the $ object of other libraries.

Here is simple way of avoiding any conflict:

// Import other Library

// Import jQuery Library

$.noConflict();

// Code that uses other library's $ can follow here.

This technique is especially effective in conjunction with the .ready() method's ability to alias the jQuery object, as within the .ready() we can use $ if we wish without fear of conflicts later:

// Import other library

// Import jQuery

$.noConflict();

jQuery(document).ready(function($) {

// Code that uses jQuery's $ can follow here.

});

// Code that uses other library's $ can follow here.