#### **Introduction**

JavaScript is the Programming Language for the Web.

## Why to study JavaScript?

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

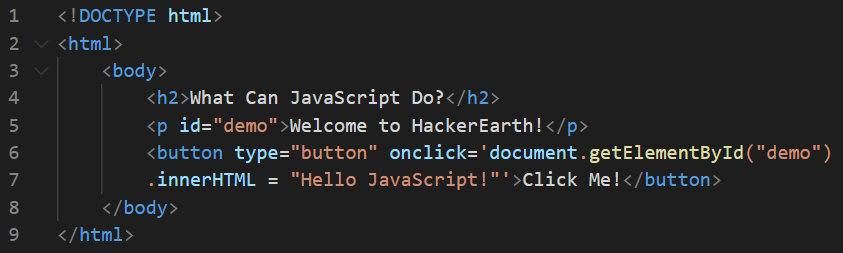
1. HTML to define the content of web pages

2. CSS to specify the layout of web pages

3. JavaScript to program the behavior of web pages

## A Simple JavaScript Document

Example



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

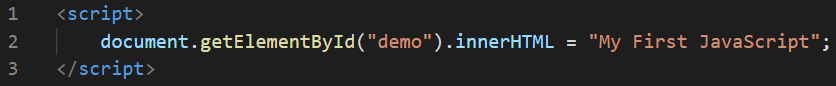
### Example Explained

* The <!DOCTYPE html> declaration defines that this document is an HTML5 document
* The <html> element is the root element of an HTML page
* The <body> element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
* The <h2> element defines a second-level heading
* The <p> element defines a paragraph
* The <button> element defines a clickable button where the JavaScript method document.getElementById() is introduced by using an onclick event.

### Properties of JavaScript

1. In HTML, JavaScript code is inserted between <script> and </script> tags. You can place any number of scripts in an HTML document.

Example



1. Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

### Advantages of JavaScript

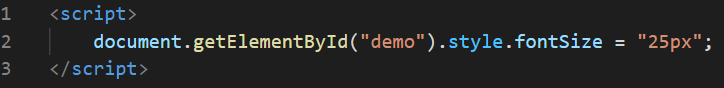
1. JavaScript accepts both double and single quotes.

## JavaScript Can Change HTML Attribute Values

## JavaScript Can Change HTML Styles (CSS):

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

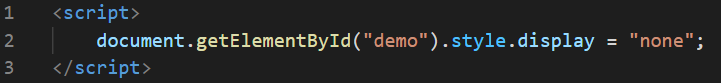
Example



## JavaScript Can Hide HTML Elements:

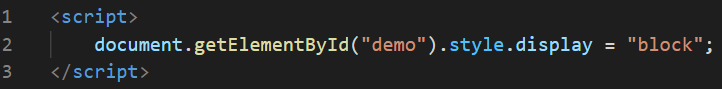
Hiding HTML elements can be done by changing the display style

Example



## JavaScript Can Show HTML Elements:

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

Example

## External JavaScript

1. External scripts are practical when the same code is used in many different web pages.
2. JavaScript files have the file extension **.js**.
3. To use an external script, put the name of the script file in the src (source) attribute of a <script> tag

Example:



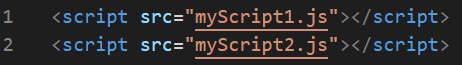
1. External scripts can be referenced with a full URL or with a path relative to the current web page.

Example:



1. Several script files can be included to one page by using several script tags at once.

Example:



## External JavaScript Advantages

Placing scripts in external files has some advantages:

1. It separates HTML and code.
2. It makes HTML and JavaScript easier to read and maintain.
3. Cached JavaScript files can speed up page loads.

## Execution of a JavaScript file

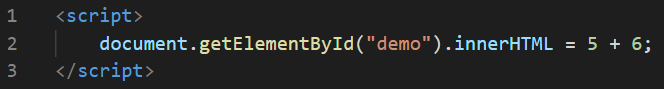
JavaScript can "display" data in different ways:

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. Writing into the browser console, using console.log().

## Various executable JavaScript methods:

## innerHTML

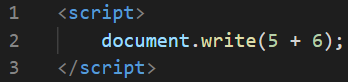
To access an HTML element, JavaScript can use the document.getElementById(id) method. The id attribute defines the HTML element. The innerHTML property defines the HTML content.  
Example



## document.write()

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

Example



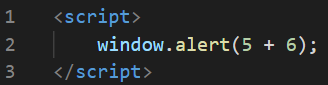
Note:

1. Using document.write() after an HTML document is loaded, will delete all existing HTML.
2. The document.write() method should only be used for testing.

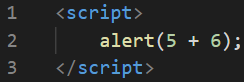
## window.alert()

You can use this method as an alert box to display data

Example



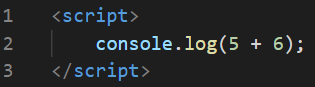
Here, you can skip the window keyword. In JavaScript, the window object is the global scope object, that means that variables, properties, and methods by default belong to the window object. This also means that specifying the window keyword is optional. Consider the following example for the better understanding:



## console.log()

For debugging purposes, you can call the console.log() method in the browser to display data.

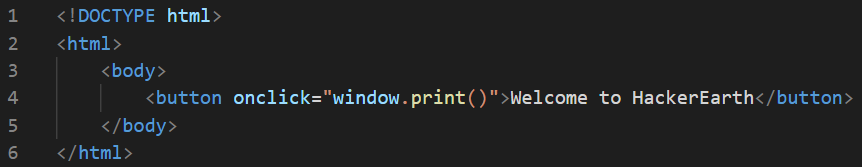
Example



## window.print()

JavaScript does not have any print object or print methods. You cannot access output devices from JavaScript. The only exception is that you can call the window.print() method in the browser to print the content of the current window.

Example

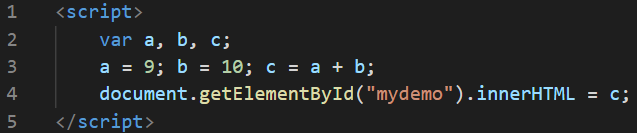


## JavaScript Statements

In a programming language, these programming instructions are called statements.

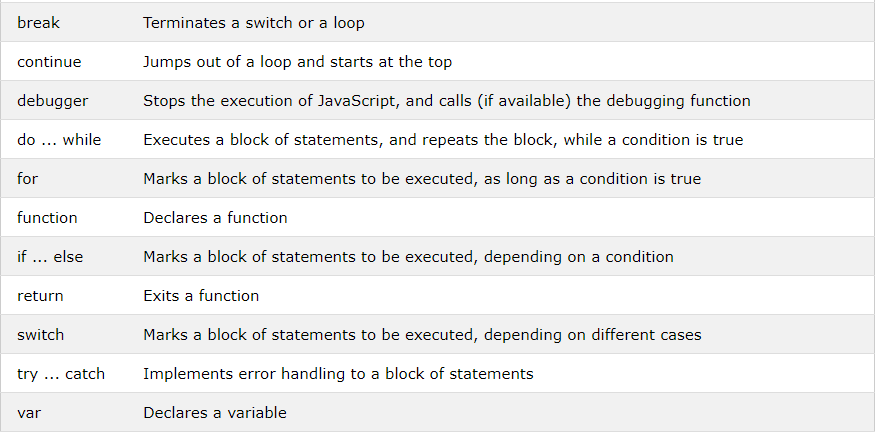
1. A JavaScript program is a list of programming statements.
2. In HTML, JavaScript programs are executed by the web browser.
3. JavaScript statements are composed of Values, Operators, Expressions, Keywords, and Comments.
4. JavaScript programs (and JavaScript statements) are often called JavaScript code.
5. Semicolons separate JavaScript statements.
6. When separated by semicolons, multiple statements on one line are allowed.

Example



1. JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.
2. JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.
3. The purpose of code blocks is to define statements to be executed together.
4. JavaScript statements often start with a keyword to identify the JavaScript action to be performed.

Here is a list of some of the keywords with its corresponding description given aside:



Note:

1. JavaScript keywords are reserved words.
2. Reserved words cannot be used as names for variables.

## JavaScript Syntax

1. The JavaScript syntax defines two types of values:
   1. Fixed values which are also called as Literals.
   2. Variable values which are also called as Variables.
2. In JavaScript, numbers can be written with or without decimals and strings are text, written within double or single quotes.
3. JavaScript uses the var keyword to declare variables and equal sign (=) is used to assign values to variables.
4. In JavaScript, double slashes (//) or /\* and \*/ are used as the single commented lines and as the multiple commented lines.
5. In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels). Here, the first character must be a letter, or an underscore (\_), or a dollar sign ($) and the subsequent characters may be letters, digits, underscores, or dollar signs. All JavaScript identifiers are case sensitive.
6. Hyphens are not allowed for variable declarations in JavaScript. They are reserved for subtractions.
7. JavaScript uses the Unicode character set.

## JavaScript Variables

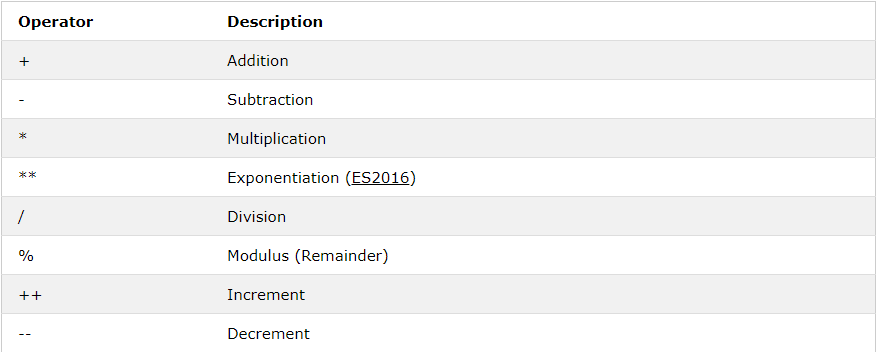
In JavaScript, we have various operators such as follows:

1. Arithmetic operators
2. Assignment operators
3. String operators
4. Comparison operators
5. Logical operators
6. Type operators
7. Bitwise operators

Arithmetic operators

Arithmetic operators perform arithmetic on numbers (literals or variables). A typical arithmetic operation operates on two numbers. The numbers (in an arithmetic operation) are called operands. The operation (to be performed between the two operands) is defined by an operator.

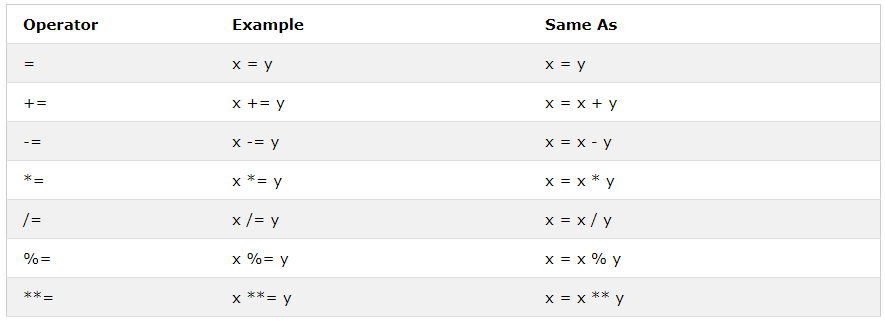
Here is the list of arithmetic operators with their corresponding description:



Assignment operators

Assignment operators assign values to JavaScript variables.

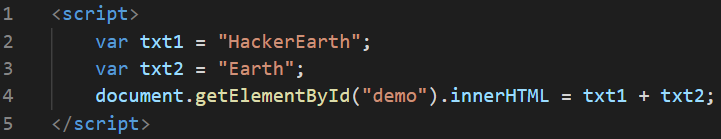
Here is the list of assignment operators with an example:



String operators

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

For example,

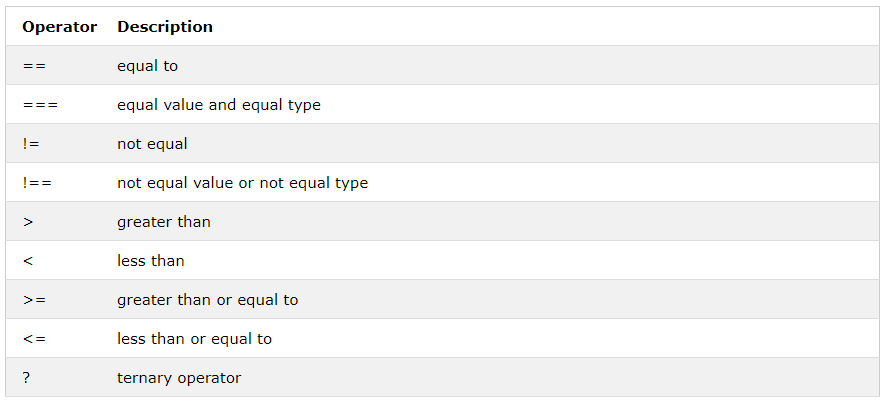


**HackerEarth** will be printed as an output for this string concatenation

Comparison operators

Comparison operators are used in logical statements to determine equality or difference between variables or values. These operations return a boolean value on applying on the conditional statements.

Here is the list of few comparison operators with their corresponding description:



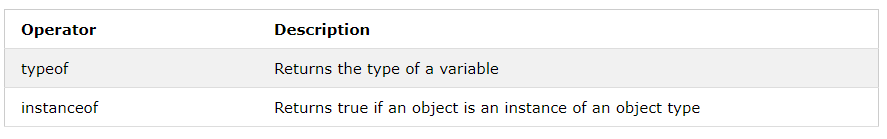
Logical operators

Logical operators are used to determine the logic between variables or values. These operations return a boolean value on applying on the conditional statements. Here is the list of logical operators with their corresponding description:



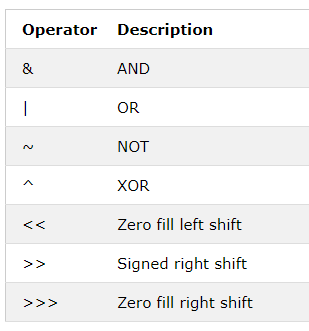
Type operators

Type operators are used to return the data type of a particular variable or to type a variable from one datatype to another datatype. Here is the list of type operators with their corresponding description:



Bitwise operators

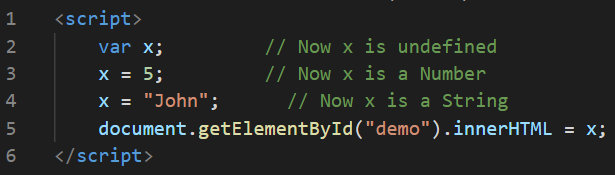
Bit operators work on 32 bits numbers. Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number. Here is the list of bitwise operators with their corresponding description:



## JavaScript Data Types

In JavaScript, we have various data types such as Number, String, object etc., JavaScript has dynamic types. This means that the same variable can be used to hold different data types.

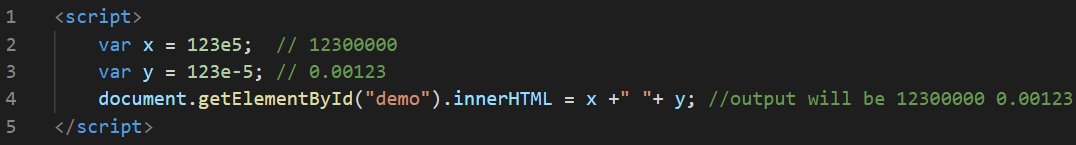
For example,



## JavaScript Numbers

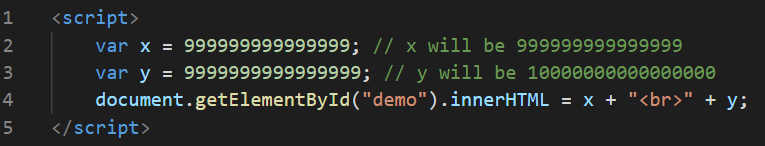
1. JavaScript has only one type of number. Numbers can be written with or without decimals.
2. Extra large or extra small numbers can be written with scientific (exponent) notation

Example:



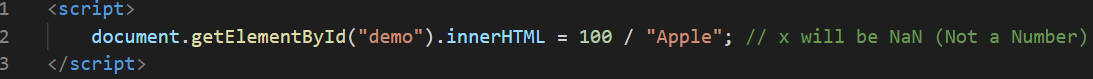
1. Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc. JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard. This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63
2. Integers (numbers without a period or exponent notation) are accurate up to 15 digits.

Example:



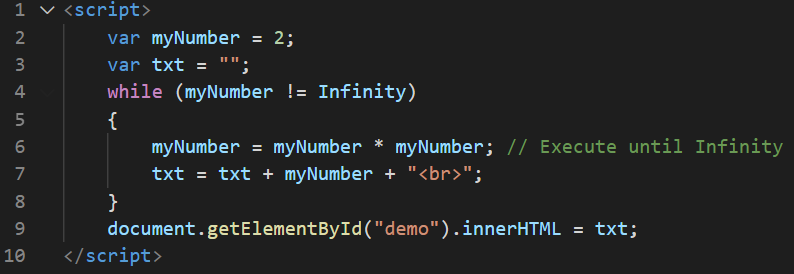
1. NaN is a JavaScript reserved word indicating that a number is not a legal number.

Example:

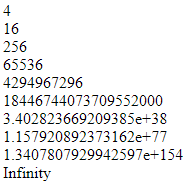


1. Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.

Example:

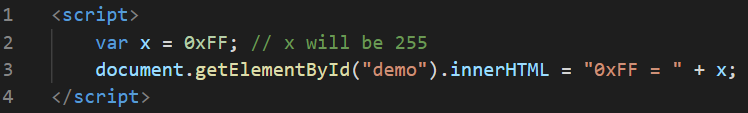


and the output will be as follows:



1. JavaScript interprets numeric constants as hexadecimal if they are preceded by 0x.

Example:



## JavaScript Number methods

Primitive values (like 3.14 or 2014), cannot have properties and methods (because they are not objects). But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

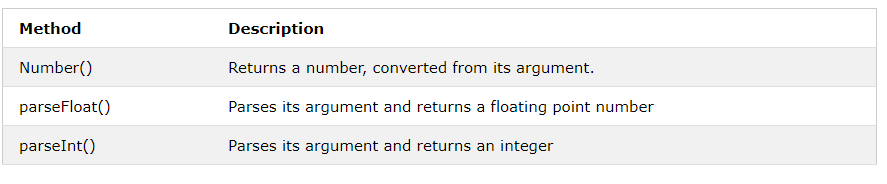
Here is the list of number methods:

1. The toString() method returns a number as a string. All number methods can be used on any type of numbers (literals, variables, or expressions).
2. The toExponential() method returns a string, with a number rounded and written using exponential notation.
3. The toFixed() method returns a string, with the number written with a specified number of decimals.
4. The toPrecision() method returns a string, with a number written with a specified length.
5. The valueOf() method returns a number as a number. In JavaScript, a number can be a primitive value (typeof = number) or an object (typeof = object). The valueOf() method is used internally in JavaScript to convert Number objects to primitive values.

## Global JavaScript methods

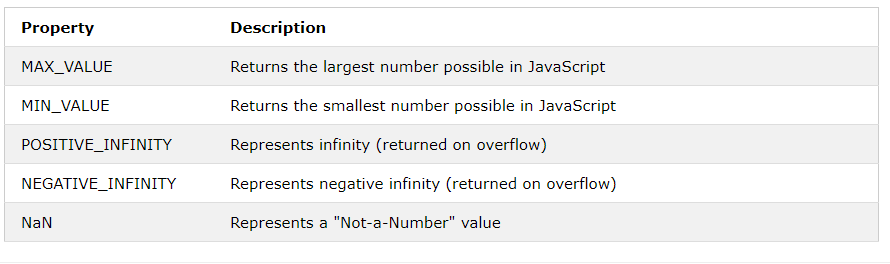
JavaScript global methods can be used on all JavaScript data types.

These are the most relevant methods, when working with numbers:



## Number properties

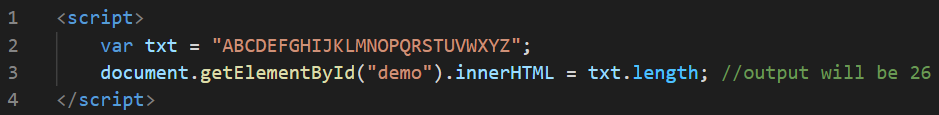
Here is the list of number properties with their corresponding descriptions:



## JavaScript Strings

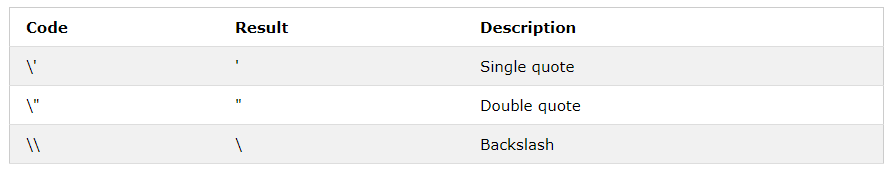
1. JavaScript strings are used for storing and manipulating text.
2. To find the length of a string, use the built-in length property.

Example:



1. The backslash (\) escape character turns special characters into string characters

Example:



## JavaScript String methods

1. The length property returns the length of a string.
2. The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string. The lastIndexOf() method returns the index of the last occurrence of a specified text in a string. Both of these methods return -1 if the text is not found.
3. The search() method searches a string for a specified value and returns the position of the match.
4. There are 3 methods for extracting a part of a string:
   1. The slice() method extracts a part of a string and returns the extracted part in a new string. This method takes 2 parameters: the start position, and the end position (end not included).
   2. The substring() method is similar to slice(). The difference is that substring() cannot accept negative indexes. If you omit the second parameter, substring() will slice out the rest of the string.
   3. The substr() method is similar to slice(). The difference is that the second parameter specifies the length of the extracted part. If you omit the second parameter, substr() will slice out the rest of the string.
5. The replace() method replaces a specified value with another value in a string. This method does not change the string it is called on. It returns a new string. By default, this method is case sensitive. To replace case insensitive, use a regular expression with an /i flag (insensitive). To replace all matches, use a regular expression with a /g flag (global match).
6. The concat() method joins two or more strings.
7. The trim() method removes whitespace from both sides of a string
8. There are 2 methods for extracting string characters:
   1. The charAt() method returns the character at a specified index (position) in a string
   2. The charCodeAt() method returns the unicode of the character at a specified index in a string
9. A string can be converted to an array with the split() methods.

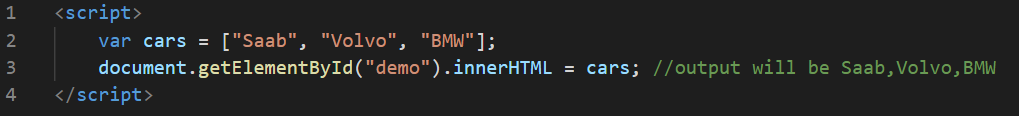
## JavaScript Arrays

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

## Creating an Array

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

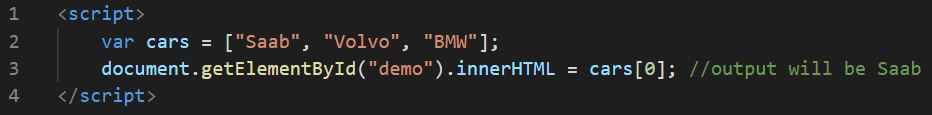
Example:



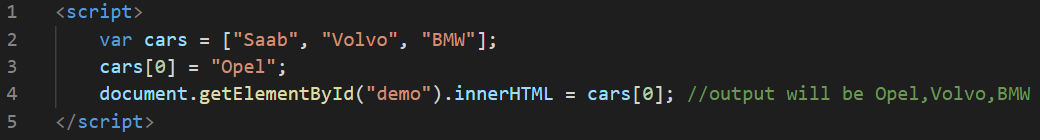
## Accessing an element from an Array

You access an array element by referring to the index number.

Example:



## Changing an Array element



## Arrays are Objects

1. Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.
2. Arrays use numbers to access its "elements".

## Array Elements Can Be Objects

1. JavaScript variables can be objects. Arrays are special kinds of objects.
2. You can have variables of different types in the same Array.
3. You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array

## The Difference Between Arrays and Objects

1. In JavaScript, arrays use numbered indexes.
2. In JavaScript, objects use named indexes.

## When to Use Arrays. When to use Objects.

1. JavaScript does not support associative arrays.
2. You should use objects when you want the element names to be strings (text).
3. You should use arrays when you want the element names to be numbers.

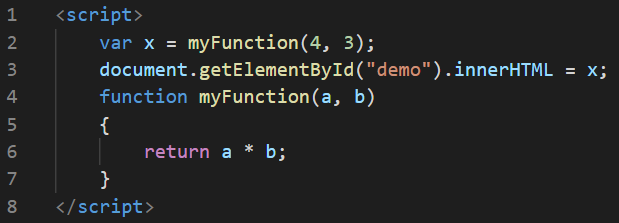
# JavaScript Array Methods

1. The JavaScript method toString() converts an array to a string of (comma separated) array values.
2. The join() method also joins all array elements into a string. It behaves just like toString(), but in addition you can specify the separator.
3. The pop() method removes the last element from an array
4. The push() method adds a new element to an array (at the end)
5. The shift() method removes the first array element and "shifts" all other elements to a lower index.
6. The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements
7. The splice() method can be used to add new items to an array
8. The concat() method creates a new array by merging (concatenating) existing arrays
9. The slice() method slices out a piece of an array into a new array.
10. JavaScript automatically converts an array to a comma separated string when a primitive value is expected.
11. The sort() method sorts an array alphabetically.
12. The reverse() method reverses the elements in an array.
13. The forEach() method calls a function (a callback function) once for each array element. This function takes 3 arguments such as the item value, the item index, and the array itself.
14. The map() method creates a new array by performing a function on each array element. It does not execute the function for array elements without values. This method does not change the original array. his function takes 3 arguments such as the item value, the item index, and the array itself.
15. The filter() method creates a new array with array elements that passes a test. This method does not change the original array. his function takes 3 arguments such as the item value, the item index, and the array itself.
16. The reduce() method runs a function on each array element to produce (reduce it to) a single value. It runs a function on each array element to produce (reduce it to) a single value.This method does not change the original array. This function takes 4 arguments such as the total, the item value, the item index, and the array itself.
17. The reduceRight() method runs a function on each array element to produce (reduce it to) a single value. It works from right-to-left in the array. This function takes 4 arguments such as the total, the item value, the item index, and the array itself.

# JavaScript Functions

JavaScript functions are defined with the function keyword.

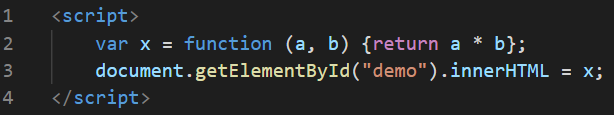
Example:



## Function Expressions

1. A JavaScript function can also be defined using an expression.
2. A function expression can be stored in a variable.
3. The function above is actually an anonymous function (a function without a name).
4. Functions stored in variables do not need function names. They are always invoked (called) using the variable name.

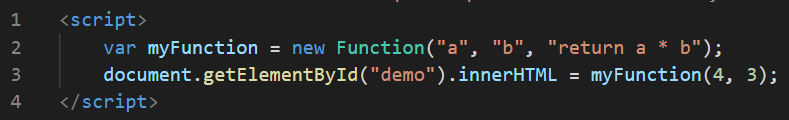
Example:



## The Function() Constructor

Functions can also be defined with a built-in JavaScript function constructor called Function().

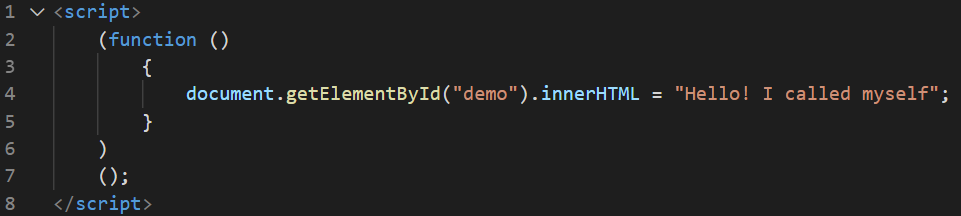
Example:



## Self-Invoking Functions

1. Function expressions can be made "self-invoking".
2. A self-invoking expression is invoked (started) automatically, without being called.
3. Function expressions will execute automatically if the expression is followed by ().
4. You cannot self-invoke a function declaration.
5. You have to add parentheses around the function to indicate that it is a function expression

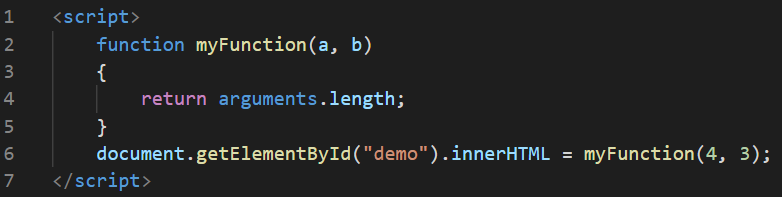
Example:



## Functions are Objects

1. The typeof operator in JavaScript returns "function" for functions.
2. But, JavaScript functions can best be described as objects.
3. JavaScript functions have both properties and methods.
4. The arguments.length property returns the number of arguments received when the function was invoked

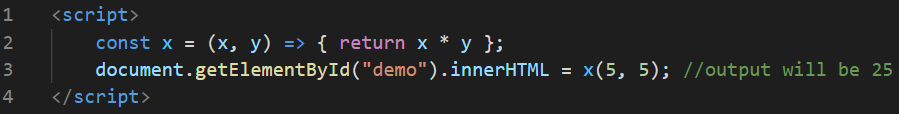
Example:



## Arrow Functions

1. Arrow functions allow a short syntaxArrow functions do not have their own this. They are not well suited for defining object methods.
2. Arrow functions are not hoisted. They must be defined before they are used.
3. Using const is safer than using var, because a function expression is always constant value.
4. You can only omit the return keyword and the curly brackets if the function is a single statement.

Example:



## Function Parameters and Arguments

A JavaScript function does not perform any checking on parameter values (arguments).

1. Function parameters are the names listed in the function definition.
2. Function arguments are the real values passed to (and received by) the function.

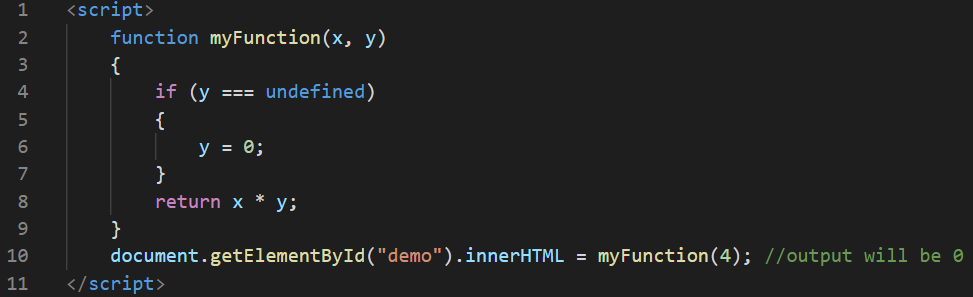
## Parameter Rules

1. JavaScript function definitions do not specify data types for parameters.
2. JavaScript functions do not perform type checking on the passed arguments.
3. JavaScript functions do not check the number of arguments received.

## Parameter Defaults

If a function is called with missing arguments (less than declared), the missing values are set to: undefined. Sometimes this is acceptable, but sometimes it is better to assign a default value to the parameter

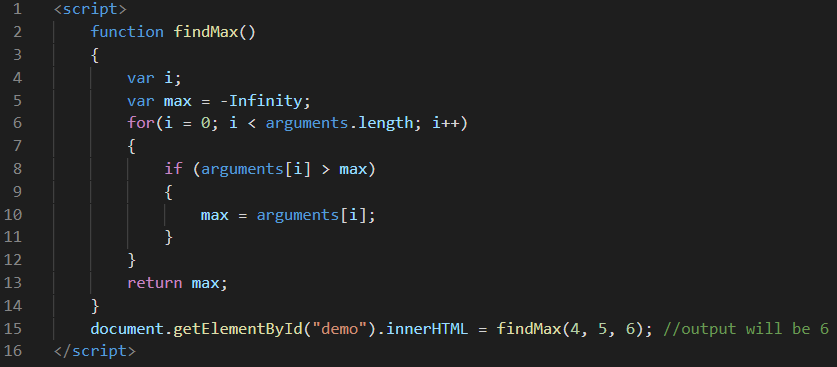
Example:



## The Arguments Object

1. JavaScript functions have a built-in object called the arguments object.
2. The argument object contains an array of the arguments used when the function was called (invoked).

Example:



## Arguments are Passed by Value

1. The parameters, in a function call, are the function's arguments.
2. JavaScript arguments are passed by value: The function only gets to know the values, not the argument's locations.
3. If a function changes an argument's value, it does not change the parameter's original value.
4. Changes to arguments are not visible (reflected) outside the function.

## Objects are Passed by Reference

1. In JavaScript, object references are values.
2. Because of this, objects will behave like they are passed by reference:
3. If a function changes an object property, it changes the original value.
4. Changes to object properties are visible (reflected) outside the function.

# JavaScript Function Call

With the call() method, you can write a method that can be used on different objects.

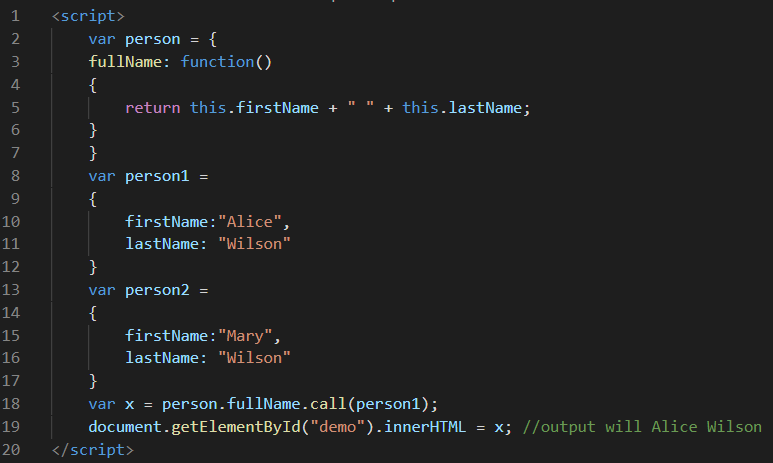
## All Functions are Methods

1. In JavaScript all functions are object methods.
2. If a function is not a method of a JavaScript object, it is a function of the global object

## The JavaScript call() Method

1. The call() method is a predefined JavaScript method.
2. It can be used to invoke (call) a method with an owner object as an argument (parameter).
3. With call(), an object can use a method belonging to another object.

Example:



1. The call() method can accept arguments

Example:



1. With the apply() method, you can write a method that can be used on different objects. The difference between call() method and apply() method is that the call() method takes arguments separately whereas the apply() method takes arguments as an array.
2. The apply() method accepts arguments in an array

Example:



1. You can find the largest number (in a list of numbers) using the Math.max() method.
2. In JavaScript strict mode, if the first argument of the apply() method is not an object, it becomes the owner (object) of the invoked function. In "non-strict" mode, it becomes the global object.

## JavaScript Closures

JavaScript variables can belong to the local or global scope.

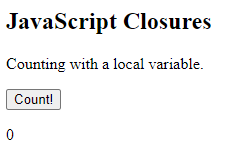
Example:



Example explained

1. The variable add is assigned the return value of a self-invoking function.
2. The self-invoking function only runs once. It sets the counter to zero (0), and returns a function expression.
3. This way add becomes a function. The "wonderful" part is that it can access the counter in the parent scope.
4. This is called a JavaScript closure. It makes it possible for a function to have "private" variables.
5. The counter is protected by the scope of the anonymous function, and can only be changed using the add function.

The output will be as follows:



Here in this example, everytime we click on the Count! button, the value below mentioned will be incremented by 1.

# JavaScript Objects

In JavaScript, almost "everything" is an object.

1. Booleans can be objects (if defined with the new keyword)
2. Numbers can be objects (if defined with the new keyword)
3. Strings can be objects (if defined with the new keyword)
4. Dates are always objects
5. Maths are always objects
6. Regular expressions are always objects
7. Arrays are always objects
8. Functions are always objects
9. Objects are always objects

All JavaScript values, except primitives, are objects.

## JavaScript Primitives

A primitive value is a value that has no properties or methods.

A primitive data type is data that has a primitive value.

JavaScript defines 5 types of primitive data types:

* string
* number
* boolean
* null
* undefined

Primitive values are immutable (they are hardcoded and therefore cannot be changed).

## Creating a JavaScript object

With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

* Define and create a single object, using an object literal.
* Define and create a single object, with the keyword new.
* Define an object constructor, and then create objects of the constructed type.

## Object Literal

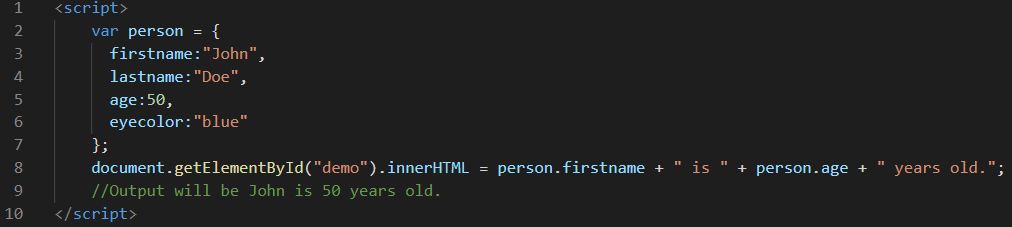
Using an object literal, you both define and create an object in one statement.

Objects are mutable: They are addressed by reference, not by value.

## Object Properties

1. Properties are the most important part of any JavaScript object.
2. Properties are the values associated with a JavaScript object.
3. A JavaScript object is a collection of unordered properties.
4. Properties can usually be changed, added, and deleted, but some are read only.

Example:



1. The delete keyword deletes a property from an object. This keyword deletes both the value of the property and the property itself. After deletion, the property cannot be used before it is added back again.
2. The delete operator is designed to be used on object properties. It has no effect on variables or functions. This operator should not be used on predefined JavaScript object properties. It can crash your application.

## Property Attributes

1. All properties have a name. In addition they also have a value.
2. The value is one of the property's attributes.
3. Other attributes are: enumerable, configurable, and writable.
4. These attributes define how the property can be accessed
5. In JavaScript, all attributes can be read, but only the value attribute can be changed (and only if the property is writable).

## Display JavaScript Objects

Some common solutions to display JavaScript objects are:

1. Displaying the Object Properties by name
2. Displaying the Object Properties in a Loop
3. Displaying the Object using Object.values()
4. Displaying the Object using JSON.stringify()

## 

## JavaScript Accessors (Getters and Setters)

Getters(get keyword) and setters (set keyword) allow you to define Object Accessors (Computed Properties).

JavaScript can secure better data quality when using getters and setters.

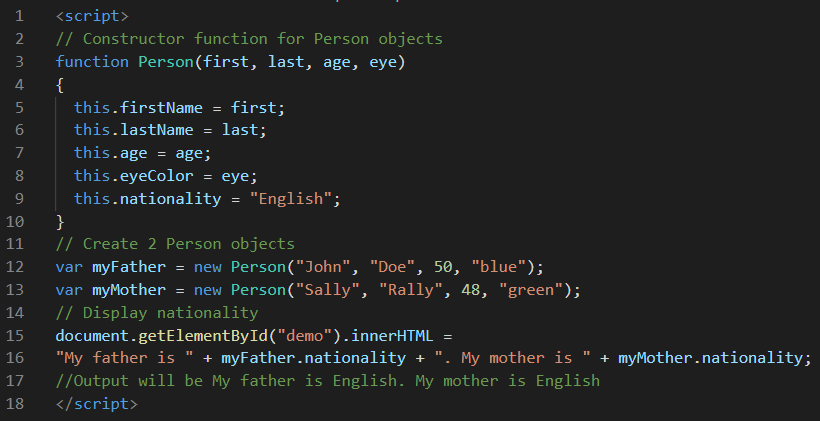
## Why Using Getters and Setters?

1. It gives simpler syntax
2. It allows equal syntax for properties and methods
3. It can secure better data quality
4. It is useful for doing things behind-the-scenes

## JavaScript Object Constructors

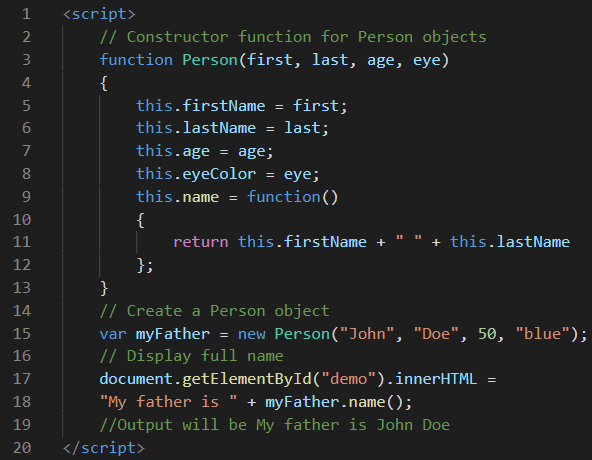
Adding a property to a constructor

Example:



## Adding a Method to a Constructor

Example



# JavaScript Object Prototypes

All JavaScript objects inherit properties and methods from a prototype.

## Prototype Inheritance

All JavaScript objects inherit properties and methods from a prototype:

1. Date objects inherit from Date.prototype
2. Array objects inherit from Array.prototype
3. Person objects inherit from Person.prototype

The Object.prototype is on the top of the prototype inheritance chain:

Date objects, Array objects, and Person objects inherit from Object.prototype.

## Adding Properties and Methods to Objects

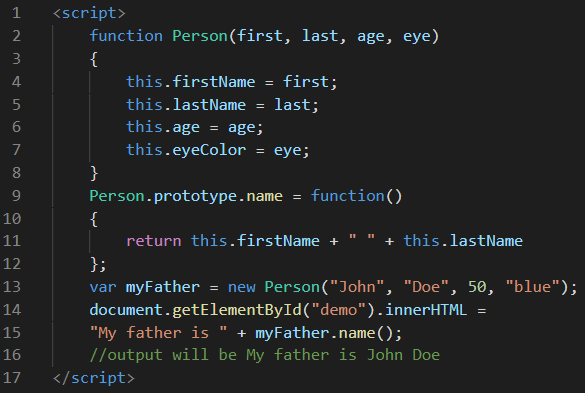
Sometimes you want to add new properties (or methods) to all existing objects of a given type.

Sometimes you want to add new properties (or methods) to an object constructor.

## Using the prototype Property

The JavaScript prototype property allows you to add new properties to object constructors.

Example:



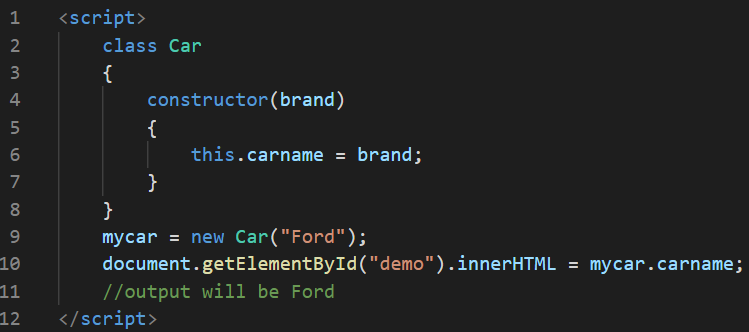
# JavaScript Classes

A class is a type of function, but instead of using the keyword function to initiate it, we use the keyword class, and the properties are assigned inside a constructor() method.

## A class definition

Use the keyword class to create a class, and always add the constructor() method. The constructor method is called each time the class object is initialized.

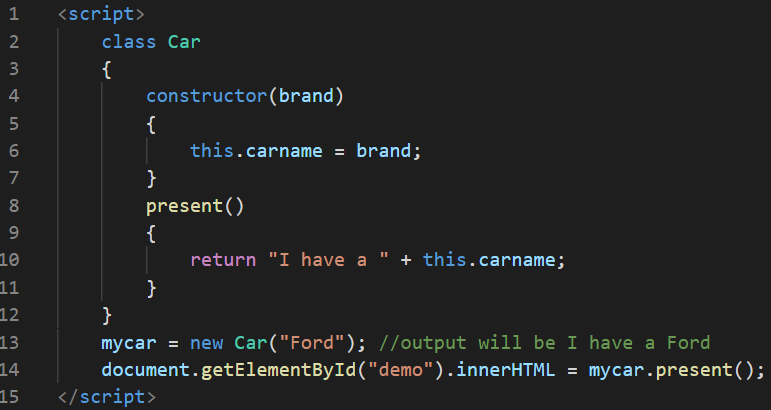
Example:



## Methods

The constructor method is special, it is where you initialize properties, it is called automatically when a class is initiated, and it has to have the exact name "constructor", in fact, if you do not have a constructor method, JavaScript will add an *invisible and empty* constructor method.

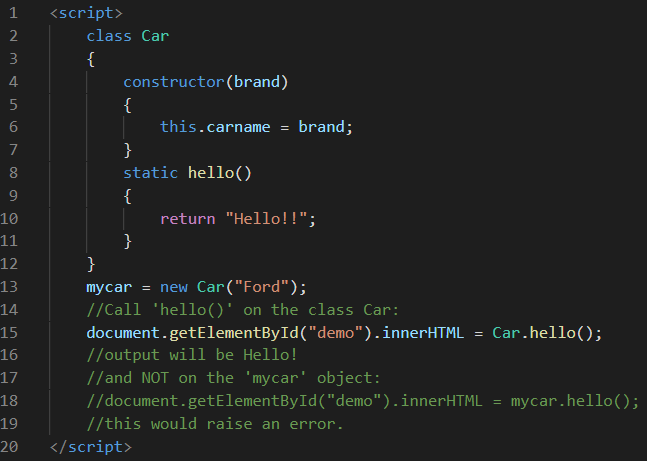
Example:



## Static Methods

Static methods are defined on the class itself, and not on the prototype.

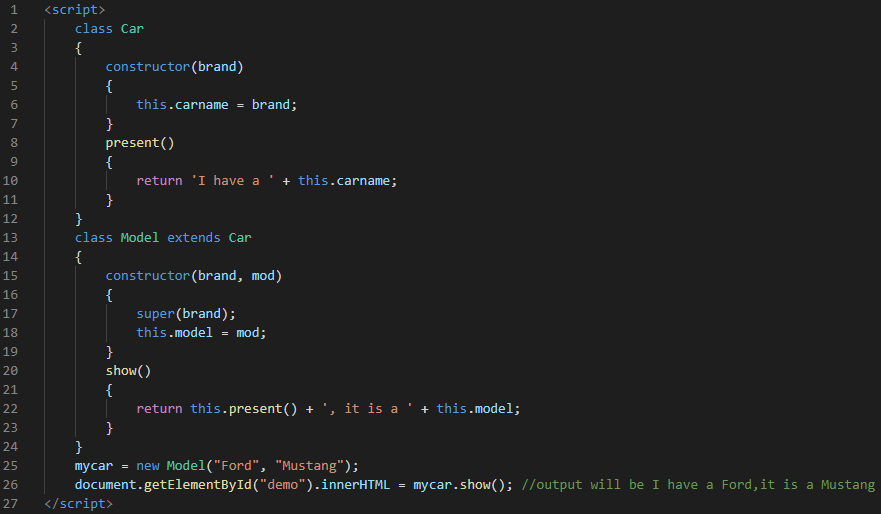
Example:



## Inheritance

To create a class inheritance, use the extends keyword. A class created with a class inheritance inherits all the methods from another class.

Example:



Here, the super() method refers to the parent class. By calling the super() method in the constructor method, we call the parent's constructor method and get access to the parent's properties and methods.

# JavaScript Events

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

Here are some examples of HTML events:

1. An HTML web page has finished loading
2. An HTML input field was changed
3. An HTML button was clicked

JavaScript lets you execute code when events are detected.

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

## What can JavaScript Do with HTML events?

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

1. Things that should be done every time a page loads
2. Things that should be done when the page is closed
3. Action that should be performed when a user clicks a button
4. Content that should be verified when a user inputs data

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

1. HTML event attributes can execute JavaScript code directly
2. HTML event attributes can call JavaScript functions
3. You can assign your own event handler functions to HTML elements
4. You can prevent events from being sent or being handled.

# JavaScript Date Objects

1. By default, JavaScript will use the browser's time zone and display a date as a full text string
2. Date objects are created with the new Date() constructor.There are 4 ways to create a new date object:
   1. new Date() creates a new date object with the current date and time
   2. new Date(*year, month, ...*) creates a new date object with a specified date and time.
   3. new Date(dateString) creates a new date object from a date string
   4. new Date(*milliseconds*) creates a new date object as zero time plus milliseconds:

## Date Methods

When a Date object is created, a number of methods allow you to operate on it. Date methods allow you to get and set the year, month, day, hour, minute, second, and millisecond of date objects, using either local time or UTC (universal, or GMT) time.

## Displaying Dates

JavaScript will (by default) output dates in full text string format

Example:

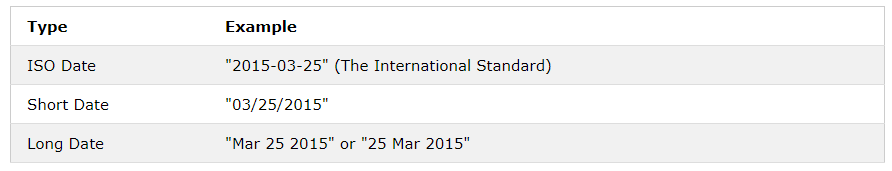


1. When you display a date object in HTML, it is automatically converted to a string, with the toString() method.
2. The toUTCString() method converts a date to a UTC string (a date display standard).
3. The toDateString() method converts a date to a more readable format.
4. The toISOString() method converts a date to a string, using the ISO standard format.

# JavaScript Date Formats

## JavaScript Date Input

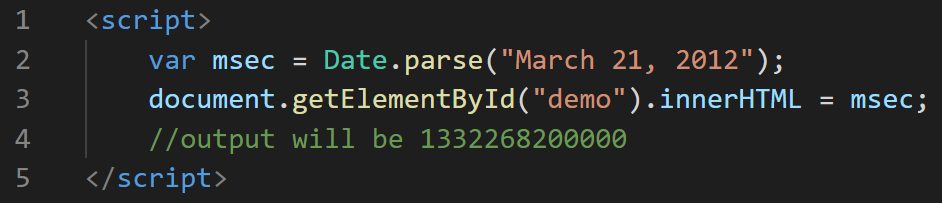
There are generally 3 types of JavaScript date input formats:



## Date Input - Parsing Dates

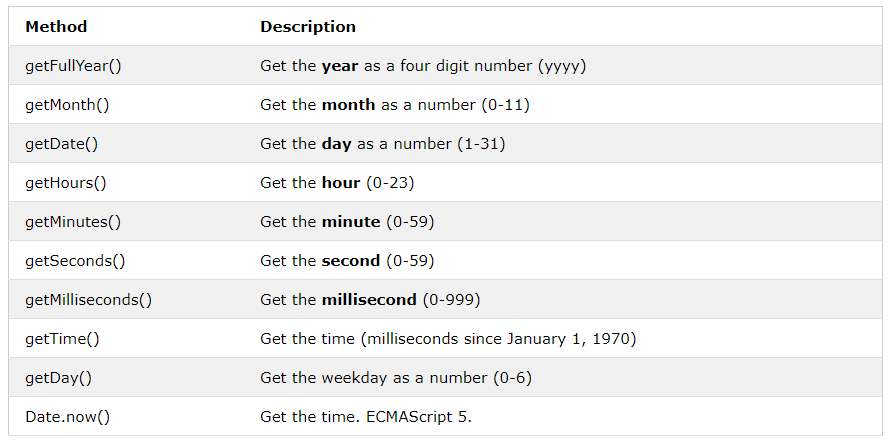
If you have a valid date string, you can use the Date.parse() method to convert it to milliseconds.

Example:



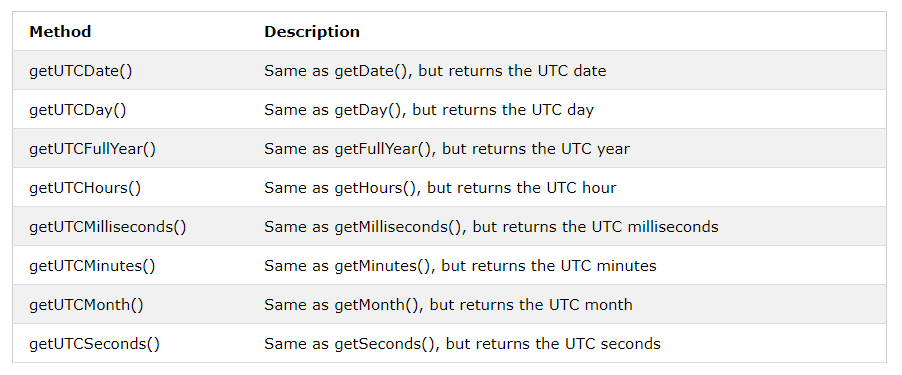
# JavaScript Get Date Methods

These methods can be used for getting information from a date object:



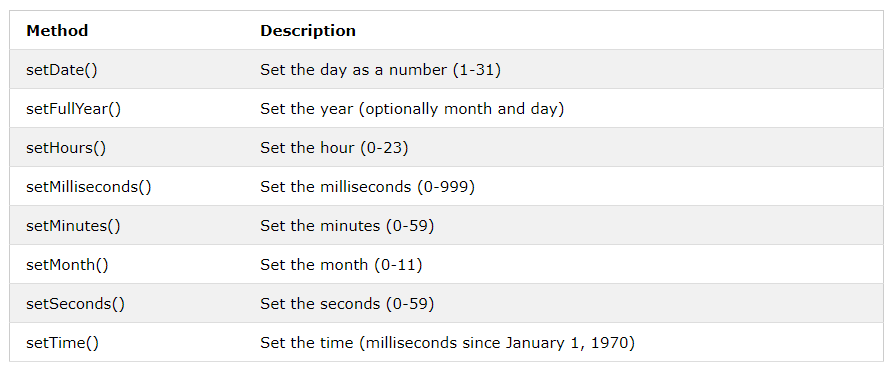
## UTC Date Methods

UTC date methods are used for working with UTC dates (Universal Time Zone dates):



## Set Date Methods

Set Date methods let you set date values (years, months, days, hours, minutes, seconds, milliseconds) for a Date Object. The following methods are used for setting a part of a date:



# JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

1. Math.round(x) returns the value of x rounded to its nearest integer
2. Math.pow(x, y) returns the value of x to the power of y
3. Math.sqrt(x) returns the square root of x
4. Math.abs(x) returns the absolute (positive) value of x
5. Math.ceil(x) returns the value of x rounded up to its nearest integer
6. Math.floor(x) returns the value of x rounded down to its nearest integer
7. Math.min() and Math.max() can be used to find the lowest or highest value in a list of arguments
8. Math.random() returns a random number between 0 (inclusive), and 1 (exclusive)

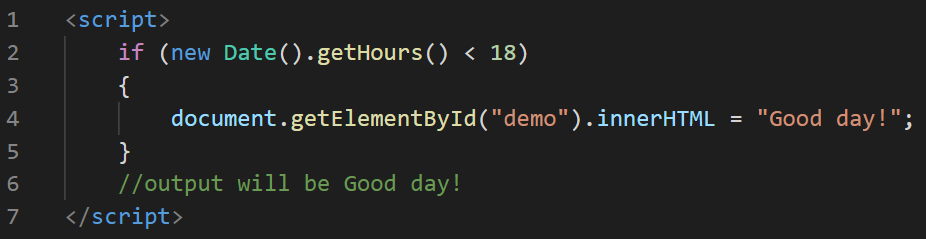
# JavaScript Conditional statements

In JavaScript we have the following conditional statements:

1. Use if to specify a block of code to be executed, if a specified condition is true
2. Use else to specify a block of code to be executed, if the same condition is false
3. Use else if to specify a new condition to test, if the first condition is false
4. Use switch to specify many alternative blocks of code to be executed.

## The if Statement

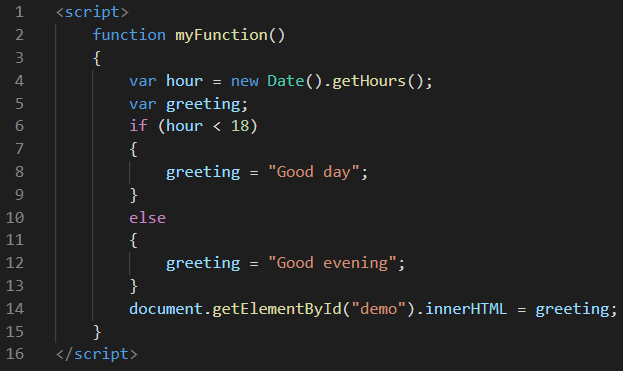
Use the if statement to specify a block of JavaScript code to be executed if a condition is true.



## The else Statement

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

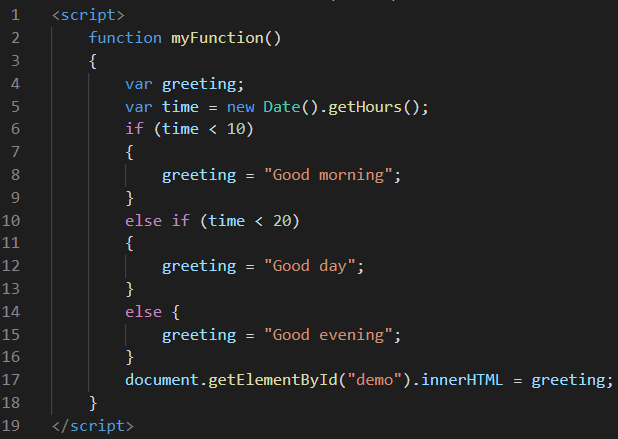
Example:



## The else if Statement

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

Example:



## The Switch Statement

The switch statement is used to select one of many code blocks to be executed.

1. The switch expression is evaluated once.
2. The value of the expression is compared with the values of each case.
3. If there is a match, the associated block of code is executed.
4. If there is no match, the default code block is executed.

## The break Keyword

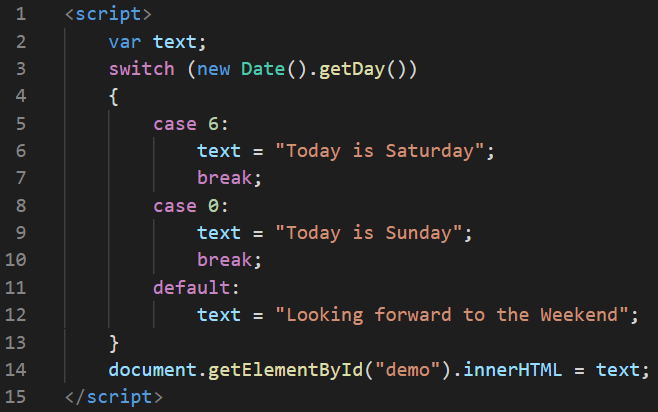
1. When JavaScript reaches a break keyword, it breaks out of the switch block.
2. This will stop the execution of inside the block.
3. It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

## The default Keyword

The default keyword specifies the code to run if there is no case match.

Here is the complete example on how a switch statement works:

Example:



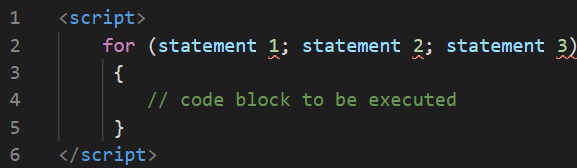
# JavaScript Loops

JavaScript supports different kinds of loops:

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

## The default Keyword

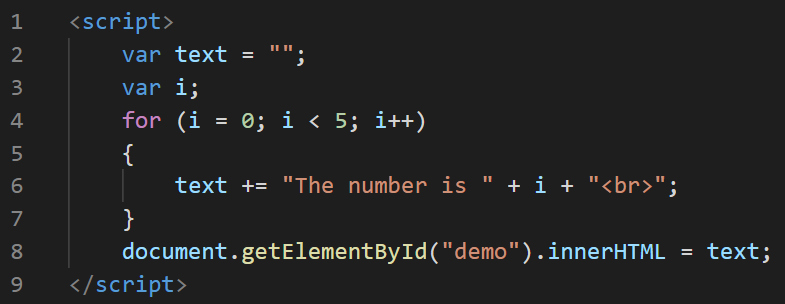
Here’s the basic syntax to implement ‘for’ loop:



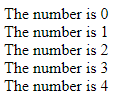
Explanation:

1. Statement 1 is executed (one time) before the execution of the code block.
2. Statement 2 defines the condition for executing the code block.
3. Statement 3 is executed (every time) after the code block has been executed.

Example:



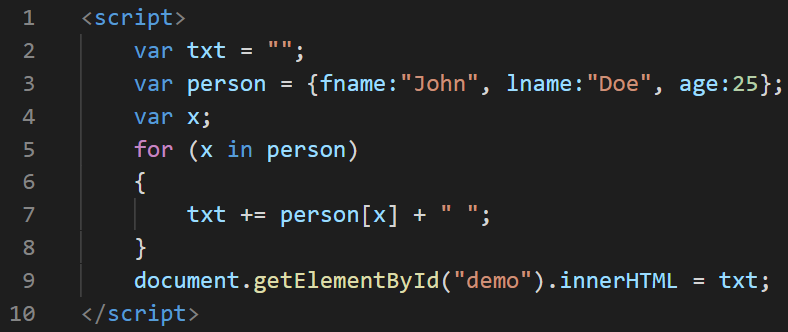
The following be the output:



## The For/In Loop

The JavaScript for/in statement loops through the properties of an object.

Example:



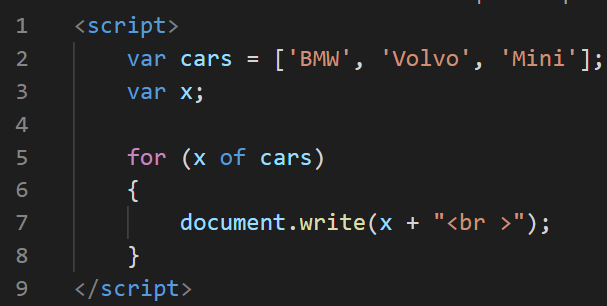
The output will be as follows:



## The For/Of Loop

The JavaScript for/of statement loops through the values of an iterable object. for/of lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more. *variable* - For every iteration the value of the next property is assigned to the variable. *Variables* can be declared with const, let, or var. *iterable* - An object that has iterable properties.

Example:



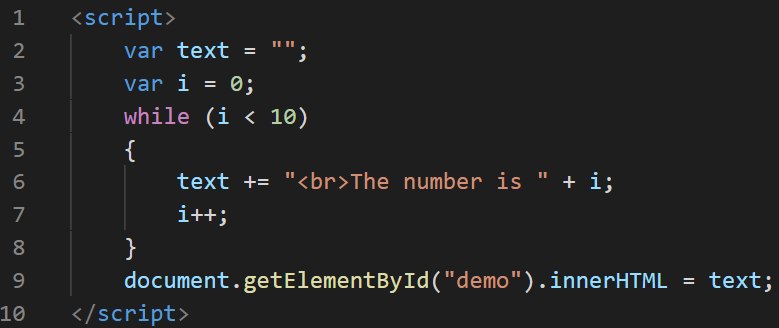
Here’s the output for the above code:



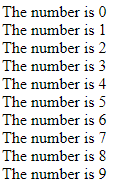
## The While Loop

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

Example:



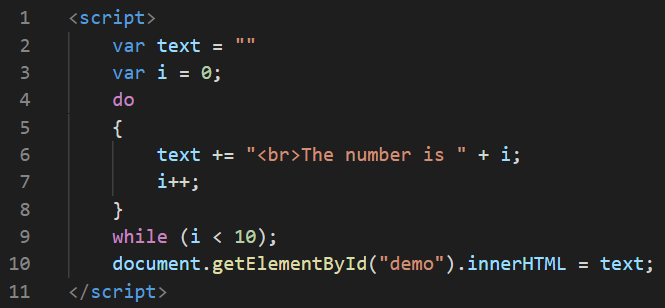
The output for the above code is follows:



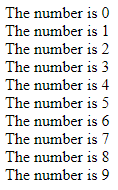
## The Do/While Loop

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

Example:



The output for the above code is as follows:



# JavaScript Regular Expressions

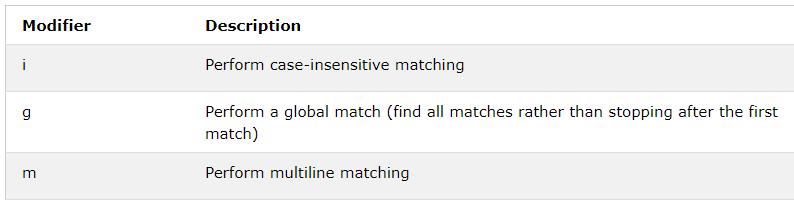
1. A regular expression is a sequence of characters that forms a search pattern.
2. When you search for data in a text, you can use this search pattern to describe what you are searching for.
3. A regular expression can be a single character, or a more complicated pattern.
4. Regular expressions can be used to perform all types of text search and text replace operations.

## String Methods for Regular Expressions

1. The search() method uses an expression to search for a match, and returns the position of the match.
2. The replace() method returns a modified string where the pattern is replaced.

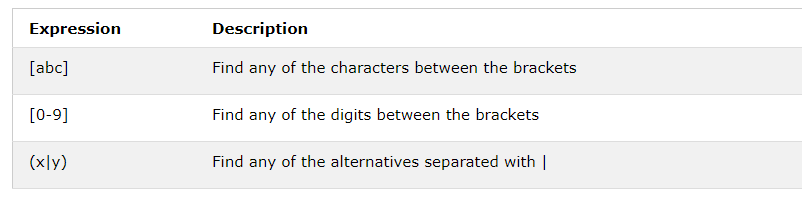
## Regular Expression Modifiers

Modifiers can be used to perform case-insensitive more global searches. Here are some Regular Expression Modifiers with their corresponding description:

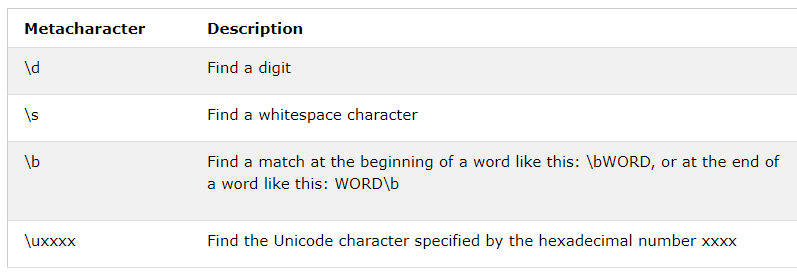


## Regular Expression Patterns

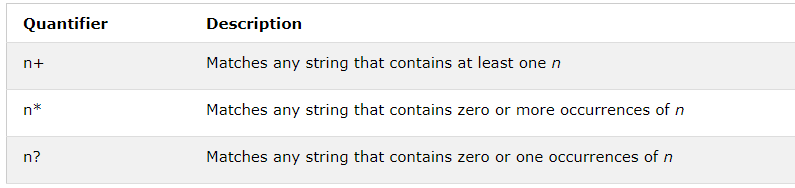
Brackets are used to find a range of characters. Here are some Regular Expression Patterns with their corresponding description:



Metacharacters are characters with a special meaning:



Quantifiers define quantities:



# JavaScript Error handling

In JavaScript, we have four types of error handling statements namely as follows:

1. The try statement lets you test a block of code for errors.
2. The catch statement lets you handle the error.
3. The throw statement lets you create custom errors.
4. The finally statement lets you execute code, after try and catch, regardless of the result.

## JavaScript try and catch

1. The try statement allows you to define a block of code to be tested for errors while it is being executed.
2. The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

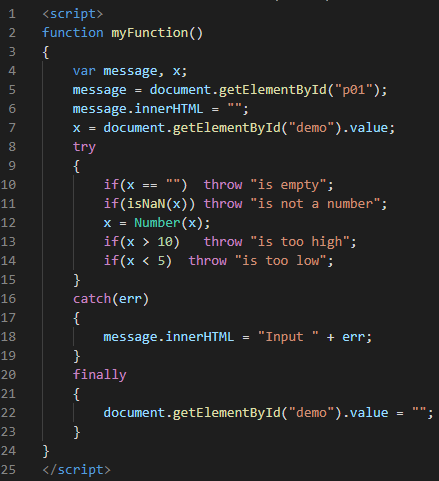
## JavaScript throws an error

1. When an error occurs, JavaScript will normally stop and generate an error message.
2. If you use throw together with try and catch, you can control program flow and generate custom error messages.

## The finally Statement

The finally statement lets you execute code, after try and catch, regardless of the result.

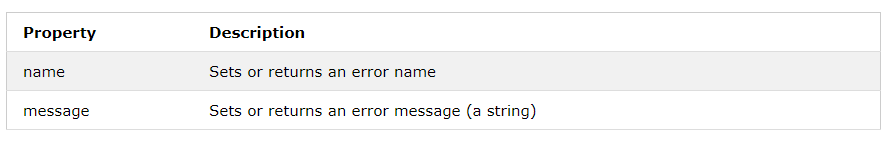
Here is the complete illustration of all the error handling statements:



## The Error Object

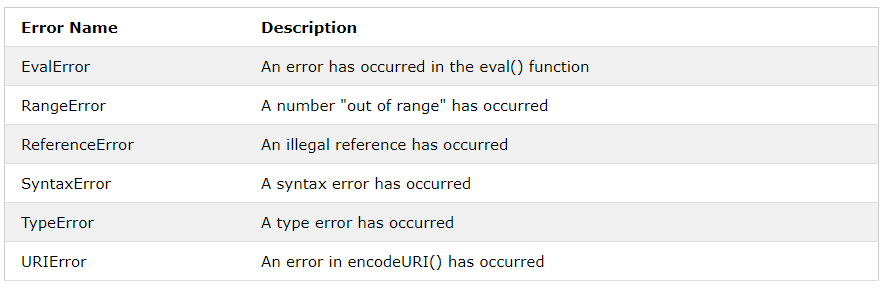
1. JavaScript has a built in error object that provides error information when an error occurs.
2. The error object provides two useful properties: name and message.

## Error Object Properties



## Error Name Values

Six different values can be returned by the error name property:



# JavaScript Debugging

## Code Debugging

1. Programming code might contain syntax errors, or logical errors.
2. Many of these errors are difficult to diagnose.
3. Often, when programming code contains errors, nothing will happen. There are no error messages, and you will get no indications where to search for errors.
4. Searching for (and fixing) errors in programming code is called code debugging.

## JavaScript Debuggers

1. Debugging is not easy. But fortunately, all modern browsers have a built-in JavaScript debugger.
2. Built-in debuggers can be turned on and off, forcing errors to be reported to the user.
3. With a debugger, you can also set breakpoints (places where code execution can be stopped), and examine variables while the code is executing.
4. Normally, otherwise follow the steps at the bottom of this page, you activate debugging in your browser with the F12 key, and select "Console" in the debugger menu.

## Setting Breakpoints

1. In the debugger window, you can set breakpoints in the JavaScript code.
2. At each breakpoint, JavaScript will stop executing, and let you examine JavaScript values.
3. After examining values, you can resume the execution of code (typically with a play button).

## The debugger Keyword

1. The debugger keyword stops the execution of JavaScript, and calls (if available) the debugging function.
2. This has the same function as setting a breakpoint in the debugger.
3. If no debugging is available, the debugger statement has no effect.
4. With the debugger turned on, this code will stop executing before it executes the third line.

# JavaScript Forms

## JavaScript Form Validation

1. HTML form validation can be done by JavaScript.
2. If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted

Example:



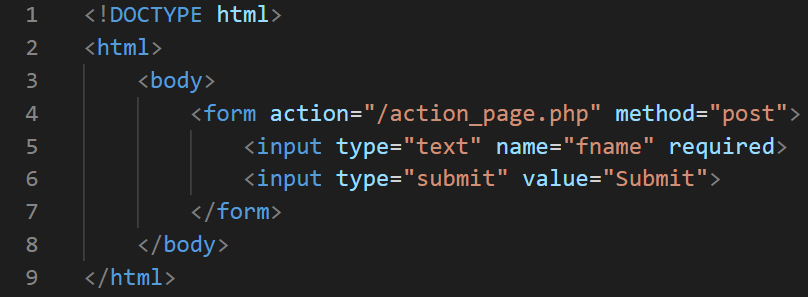
This is how the output of this code looks like:



## Automatic HTML Form Validation

1. HTML form validation can be performed automatically by the browser
2. If a form field (fname) is empty, the required attribute prevents this form from being submitted.

Example:



This is how the output will be if you have clicked the button with an empty text field:



## Data Validation

Data validation is the process of ensuring that user input is clean, correct, and useful.

Validation can be defined by many different methods, and deployed in many different ways.

1. Server side validation is performed by a web server, after input has been sent to the server.
2. Client side validation is performed by a web browser, before input is sent to a web server.

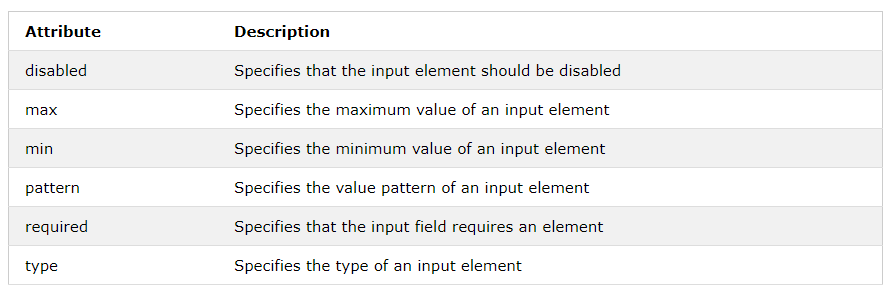
## HTML Constraint Validation

HTML5 introduced a new HTML validation concept called constraint validation.

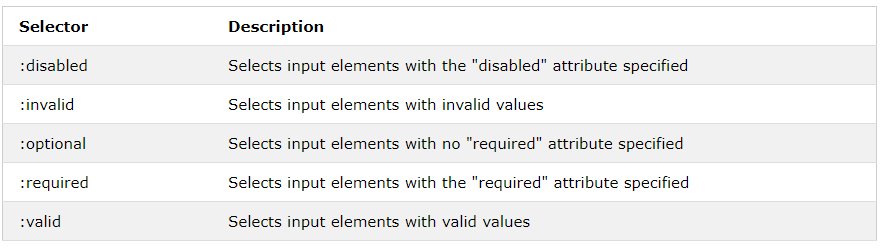
HTML constraint validation is based on:

1. Constraint validation HTML Input Attributes.
2. Constraint validation CSS Pseudo Selectors.
3. Constraint validation DOM Properties and Methods.

## Constraint Validation HTML Input Attributes

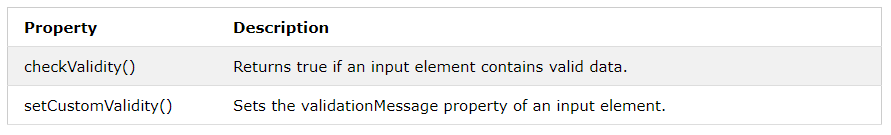


## Constraint Validation CSS Pseudo Selectors

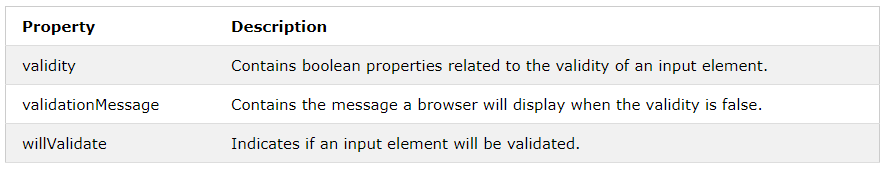


# JavaScript Validation API

## Constraint Validation DOM Methods

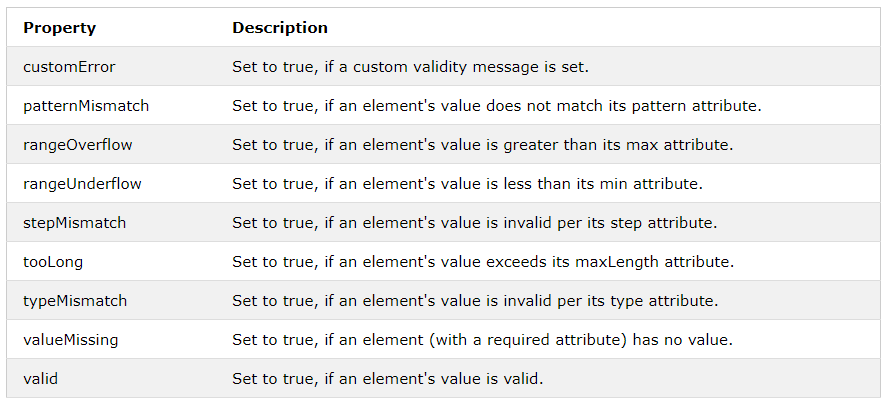


## Constraint Validation DOM Properties



## Validity Properties

The validity property of an input element contains a number of properties related to the validity of data:



# JavaScript HTML DOM

When a web page is loaded, the browser creates a Document Object Model of the page.

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

1. JavaScript can change all the HTML elements in the page
2. JavaScript can change all the HTML attributes in the page
3. JavaScript can change all the CSS styles in the page
4. JavaScript can remove existing HTML elements and attributes
5. JavaScript can add new HTML elements and attributes
6. JavaScript can react to all existing HTML events in the page
7. JavaScript can create new HTML events in the page

## What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.

The W3C DOM standard is separated into 3 different parts:

1. Core DOM - standard model for all document types
2. XML DOM - standard model for XML documents
3. HTML DOM - standard model for HTML documents

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

1. The HTML elements as objects
2. The properties of all HTML elements
3. The methods to access all HTML elements
4. The events for all HTML elements

# JavaScript - HTML DOM Methods

1. HTML DOM methods are actions you can perform (on HTML Elements).
2. HTML DOM properties are values (of HTML Elements) that you can set or change.

## The DOM Programming Interface

1. The HTML DOM can be accessed with JavaScript (and with other programming languages).
2. In the DOM, all HTML elements are defined as objects.
3. The programming interface is the properties and methods of each object.
4. A property is a value that you can get or set (like changing the content of an HTML element).
5. A method is an action you can do (like add or deleting an HTML element).

Example:



The output of this above code is as follows:



Explanation for the example:

1. In the example above the getElementById method used id="demo" to find the element.
2. The innerHTML property is useful for getting or replacing the content of HTML elements.

# JavaScript HTML DOM Document

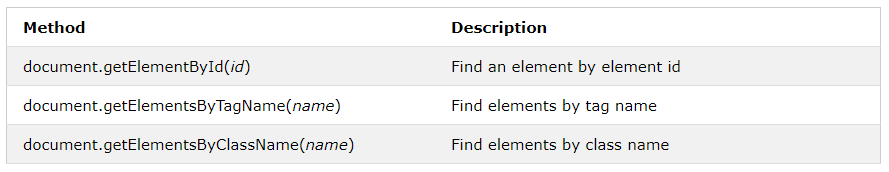
The HTML DOM document object is the owner of all other objects in your web page.

## The HTML DOM Document Object

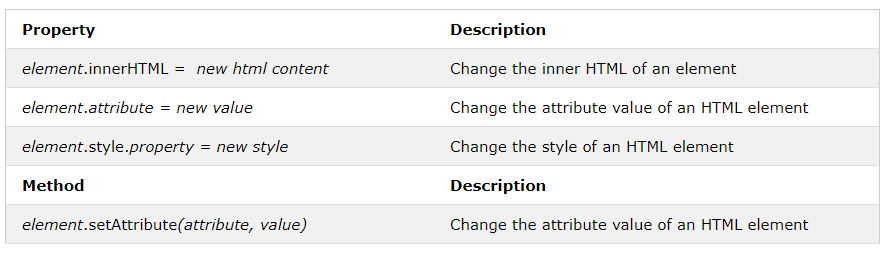
1. The document object represents your web page.
2. If you want to access any element in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

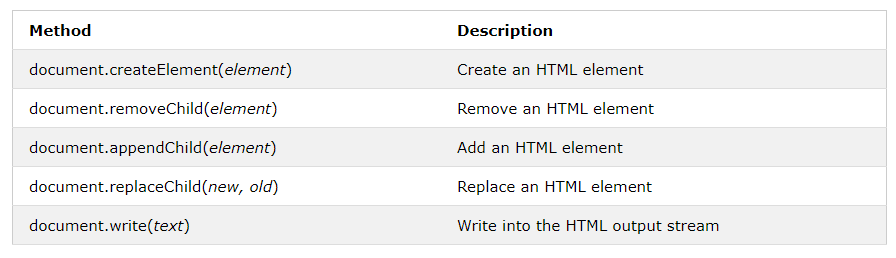
## Finding HTML Elements



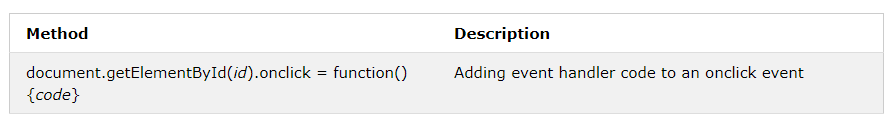
## Changing HTML Elements



## Adding and Deleting Elements



## Adding Events Handlers



## Finding HTML Elements by CSS Selectors

If you want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

## Events

The HTML DOM allows you to execute code when an event occurs.

Events are generated by the browser when "things happen" to HTML elements:

* An element is clicked on
* The page has loaded
* Input fields are changed

Here is the list of more DOM events with their corresponding descriptions:

1. The onload and onunload events are triggered when the user enters or leaves the page.
2. The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.
3. The onload and onunload events can be used to deal with cookies.
4. The onchange event is often used in combination with validation of input fields.
5. The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element.
6. The onmousedown, onmouseup, and onclick events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

# JavaScript HTML DOM Animation

## Create an Animation Container

All animations should be relative to a container element.

Example:



## Animation Code

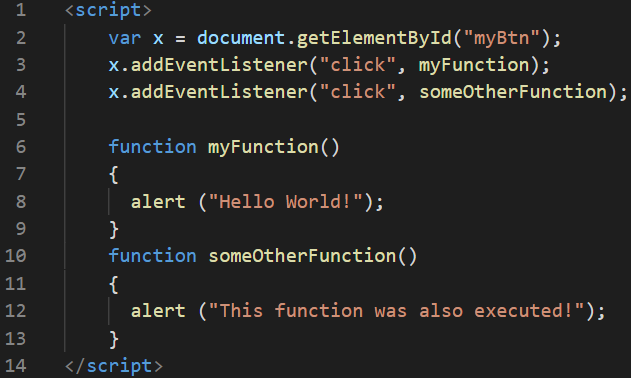
1. JavaScript animations are done by programming gradual changes in an element's style.
2. The changes are called by a timer. When the timer interval is small, the animation looks continuous.

# JavaScript HTML DOM EventListener

## The addEventListener() method

1. The addEventListener() method attaches an event handler to the specified element.
2. The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.

Example:



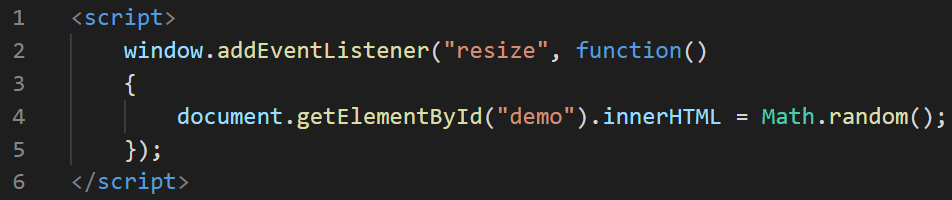
1. You can add many event handlers to one element.

Example:



1. You can add many event handlers of the same type to one element, i.e two "click" events.
2. You can add event listeners to any DOM object, not only HTML elements. i.e the window object.

Example:



1. The addEventListener() method makes it easier to control how the event reacts to bubbling.
2. When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.
3. You can easily remove an event listener by using the removeEventListener() method.

## Event Bubbling or Event Capturing?

There are two ways of event propagation in the HTML DOM, bubbling and capturing.

1. In *bubbling* the innermost element's event is handled first and then the outer: the <p> element's click event is handled first, then the <div> element's click event.
2. In *capturing* the outermost element's event is handled first and then the inner: the <div> element's click event will be handled first, then the <p> element's click event.

## The removeEventListener() method

The removeEventListener() method removes event handlers that have been attached with the addEventListener() method.

# JavaScript HTML DOM Navigation

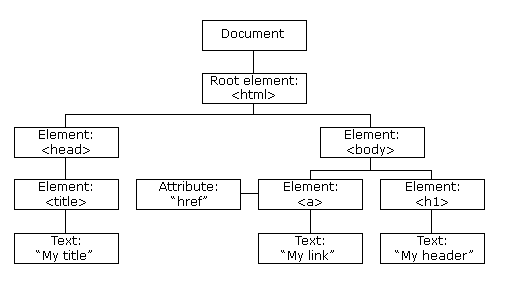
With the HTML DOM, you can navigate the node tree using node relationships.

## DOM Nodes

According to the W3C HTML DOM standard, everything in an HTML document is a node:

1. The entire document is a document node.
2. Every HTML element is an element node.
3. The text inside HTML elements are text nodes.
4. Every HTML attribute is an attribute node (deprecated).
5. All comments are comment nodes.
6. With the HTML DOM, all nodes in the node tree can be accessed by JavaScript.
7. New nodes can be created, and all nodes can be modified or deleted.

Here is the sample tree diagram of the document node:



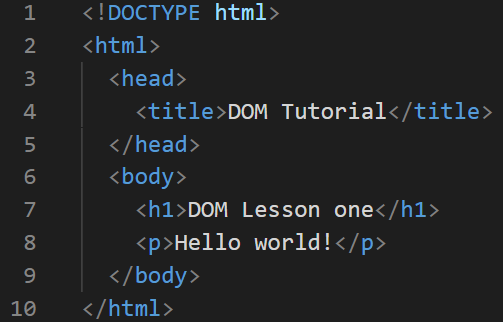
## Node Relationships

The nodes in the node tree have a hierarchical relationship to each other.

The terms parent, child, and sibling are used to describe the relationships.

1. In a node tree, the top node is called the root (or root node)
2. Every node has exactly one parent, except the root (which has no parent)
3. A node can have a number of children
4. Siblings (brothers or sisters) are nodes with the same parent.

Example:



From the HTML above you can read:

1. <html> is the root node
2. <html> has no parents
3. <html> is the parent of <head> and <body>
4. <head> is the first child of <html>
5. <body> is the last child of <html>

and:

1. <head> has one child: <title>
2. <title> has one child (a text node): "DOM Tutorial"
3. <body> has two children: <h1> and <p>
4. <h1> has one child: "DOM Lesson one"
5. <p> has one child: "Hello world!"
6. <h1> and <p> are siblings

## Navigating Between Nodes

You can use the following node properties to navigate between nodes with JavaScript:

1. parentNode
2. childNodes[*nodenumber*]
3. firstChild
4. lastChild
5. nextSibling
6. previousSibling

## DOM Root Nodes

There are two special properties that allow access to the full document:

1. document.body - The body of the document
2. document.documentElement - The full document

## The nodeName Property

The nodeName property specifies the name of a node.

1. nodeName is read-only
2. nodeName of an element node is the same as the tag name
3. nodeName of an attribute node is the attribute name
4. nodeName of a text node is always #text
5. nodeName of the document node is always #document

## The nodeValue Property

The nodeValue property specifies the value of a node.

1. nodeValue for element nodes is null
2. nodeValue for text nodes is the text itself
3. nodeValue for attribute nodes is the attribute value

## The nodeType Property

The nodeType property is read only. It returns the type of a node.

# JavaScript HTML DOM Collections

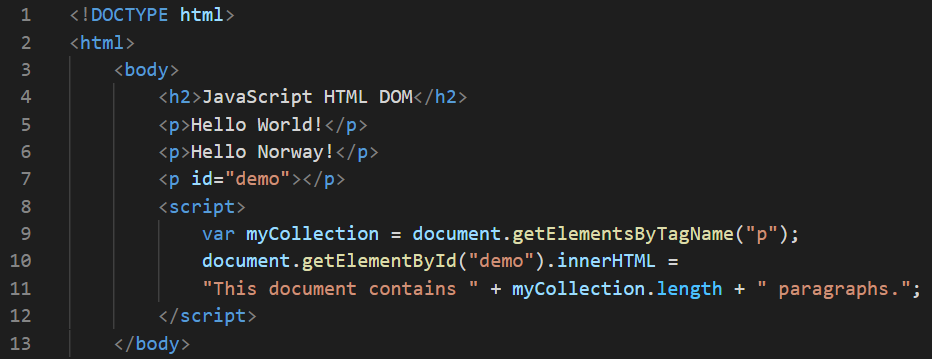
## The HTMLCollection Object

1. The getElementsByTagName() method returns an HTMLCollection object.
2. An HTMLCollection object is an array-like list (collection) of HTML elements.

## HTML HTMLCollection Length

The length property defines the number of elements in an HTMLCollection:

Example:



Example explained:

1. Create a collection of all <p> elements
2. Display the length of the collection

# JavaScript HTML DOM Node Lists

## The HTML DOM NodeList Object

1. A NodeList object is a list (collection) of nodes extracted from a document.
2. A NodeList object is almost the same as an HTMLCollection object.
3. Some (older) browsers return a NodeList object instead of an HTMLCollection for methods like getElementsByClassName().
4. All browsers return a NodeList object for the property childNodes.

## The Difference Between an HTMLCollection and a NodeList

1. An HTMLCollection (previous chapter) is a collection of HTML elements.
2. A NodeList is a collection of document nodes.
3. A NodeList and an HTML collection are very much the same thing.
4. Both an HTMLCollection object and a NodeList object is an array-like list (collection) of objects.
5. Both have a length property defining the number of items in the list (collection).
6. Both provide an index (0, 1, 2, 3, 4, ...) to access each item like an array.
7. HTMLCollection items can be accessed by their name, id, or index number.
8. NodeList items can only be accessed by their index number.
9. Only the NodeList object can contain attribute nodes and text nodes.

# JavaScript Window - The Browser Object Model (BOM)

The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.

## The Browser Object Model (BOM)

1. There are no official standards for the Browser Object Model (BOM).
2. Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity, it is often referred to as methods and properties of the BOM.

## The Window Object

1. The window object is supported by all browsers. It represents the browser's window.
2. All global JavaScript objects, functions, and variables automatically become members of the window object.
3. Global variables are properties of the window object.
4. Global functions are methods of the window object.
5. Even the document object (of the HTML DOM) is a property of the window object.

## Window Size

Two properties can be used to determine the size of the browser window.

Both properties return the sizes in pixels:

1. window.innerHeight - the inner height of the browser window (in pixels).
2. window.innerWidth - the inner width of the browser window (in pixels).

## Other Window Methods

Some other methods:

* window.open() - open a new window
* window.close() - close the current window
* window.moveTo() - move the current window
* window.resizeTo() - resize the current window

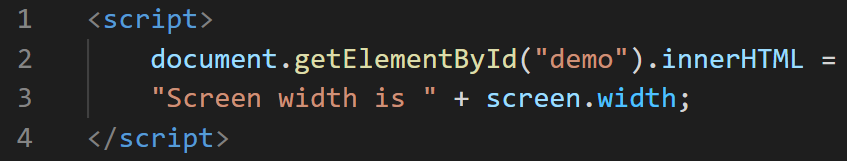
# JavaScript Window Screen

The window.screen object can be written without the window prefix.

Properties:

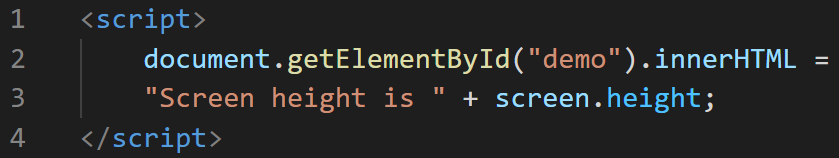
1. The screen.width property returns the width of the visitor's screen in pixels.

Example:



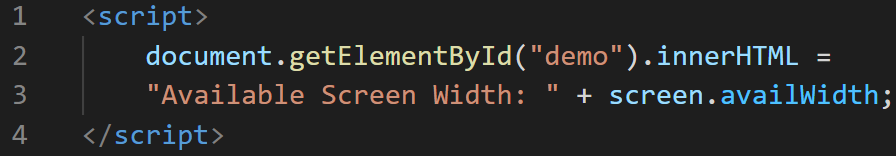
1. The screen.height property returns the height of the visitor's screen in pixels.

Example:



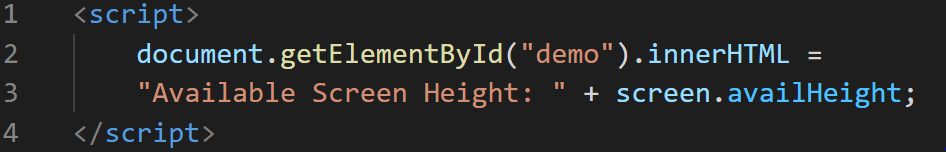
1. The screen.availWidth property returns the width of the visitor's screen, in pixels, minus interface features like the Windows Taskbar.

Example:



1. The screen.availHeight property returns the height of the visitor's screen, in pixels, minus interface features like the Windows Taskbar.

Example:



1. The screen.colorDepth property returns the number of bits used to display one color.

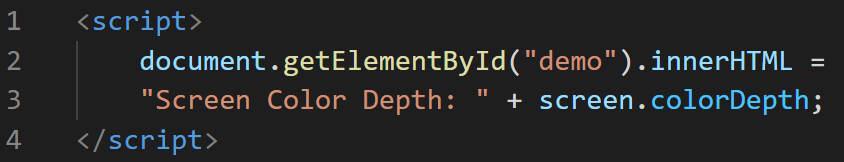
All modern computers use 24 bit or 32 bit hardware for color resolution:

* 24 bits = 16,777,216 different "True Colors"
* 32 bits = 4,294,967,296 different "Deep Colors"

Older computers used 16 bits: 65,536 different "High Colors" resolution.

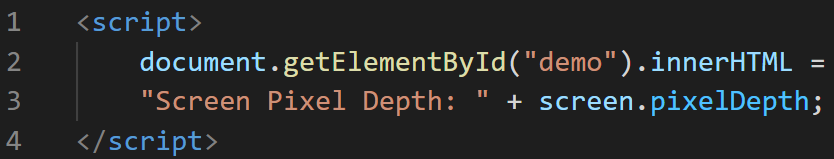
Very old computers, and old cell phones used 8 bits: 256 different "VGA colors"

Example:



The screen.pixelDepth property returns the pixel depth of the screen.

Example:



# JavaScript Window Location

The window.location object can be used to get the current page address (URL) and to redirect the browser to a new page.

## Window Location

The window.location object can be written without the window prefix.

Some examples:

1. window.location.href returns the href (URL) of the current page
2. window.location.hostname returns the domain name of the web host
3. window.location.pathname returns the path and filename of the current page
4. window.location.protocol returns the web protocol used (http: or https:)
5. window.location.assign() loads a new document.

# JavaScript Window History

The window.history object contains the browser's history.

## Window History

The window.history object can be written without the window prefix.

To protect the privacy of the users, there are limitations to how JavaScript can access this object.

Some methods:

1. history.back() - same as clicking back in the browser.
2. history.forward() - same as clicking forward in the browser.

# JavaScript Window Navigator

The window.navigator object contains information about the visitor's browser.

## Window Navigator

The window.navigator object can be written without the window prefix.

Some examples:

1. navigator.appName
2. navigator.appCodeName
3. navigator.platform

## 

## Browser Cookies

1. The cookieEnabled property returns true if cookies are enabled, otherwise false.
2. The appName property returns the application name of the browser.
3. The appCodeName property returns the application code name of the browser.
4. The product property returns the product name of the browser engine.
5. The appVersion property returns version information about the browser.
6. The userAgent property returns the user-agent header sent by the browser to the server.
7. The platform property returns the browser platform (operating system).
8. The language property returns the browser's language
9. The onLine property returns true if the browser is online.
10. The javaEnabled() method returns true if Java is enabled.

# JavaScript Popup Alerts

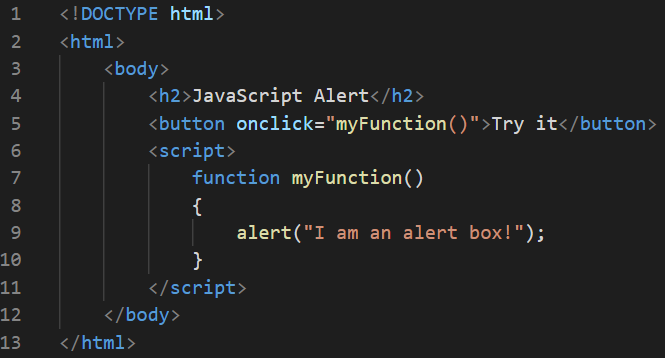
JavaScript has three kinds of popup boxes: Alert box, Confirm box, and Prompt box.

## Alert Box

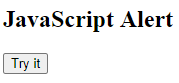
An alert box is often used if you want to make sure information comes through to the user.

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

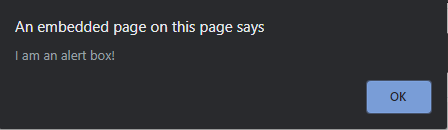
Example:



This is how the output of the above HTML code looks like:



On clicking the “Try it” button,



## Confirm Box

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

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

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

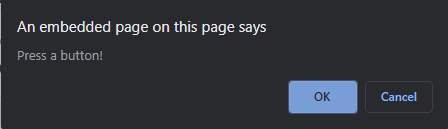
Example:



This is how the output of the above HTML code looks like:



On clicking the “Try it” button,



## Prompt Box

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

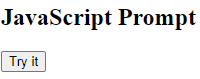
When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

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

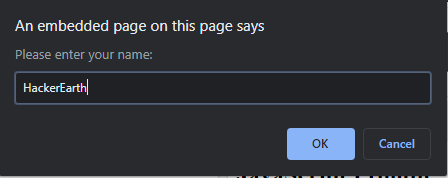
Example:



This is how the output of the above HTML code looks like:



On clicking the “Try it” button,



# JavaScript AJAX Introduction

## What is AJAX?

AJAX = Asynchronous JavaScript And XML.

AJAX is not a programming language.

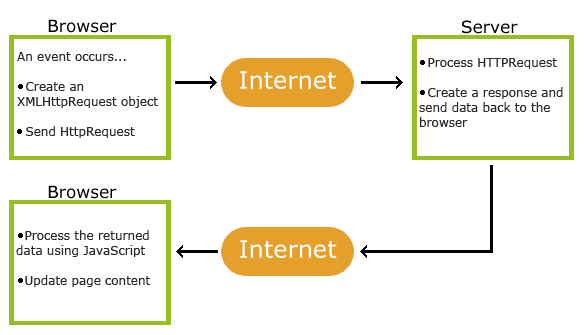
AJAX just uses a combination of:

1. A browser built-in XMLHttpRequest object (to request data from a web server)
2. JavaScript and HTML DOM (to display or use the data)

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

## How AJAX Works

Here is the flow of working of AJAX:



The following are the stages on the working flow of AJAX:

1. An event occurs in a web page (the page is loaded, a button is clicked).
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server.
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript

# AJAX - The XMLHttpRequest Object

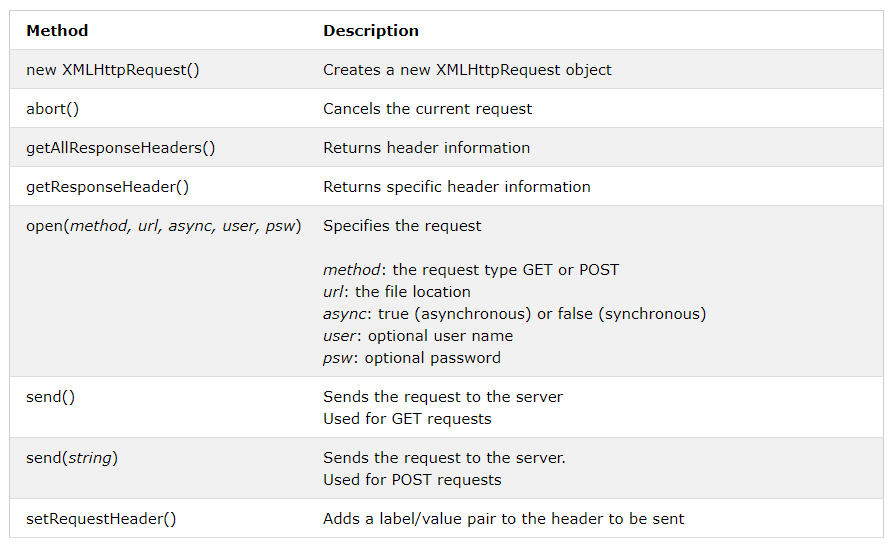
The keystone of AJAX is the XMLHttpRequest object.

## The XMLHttpRequest Object

All modern browsers support the XMLHttpRequest object.

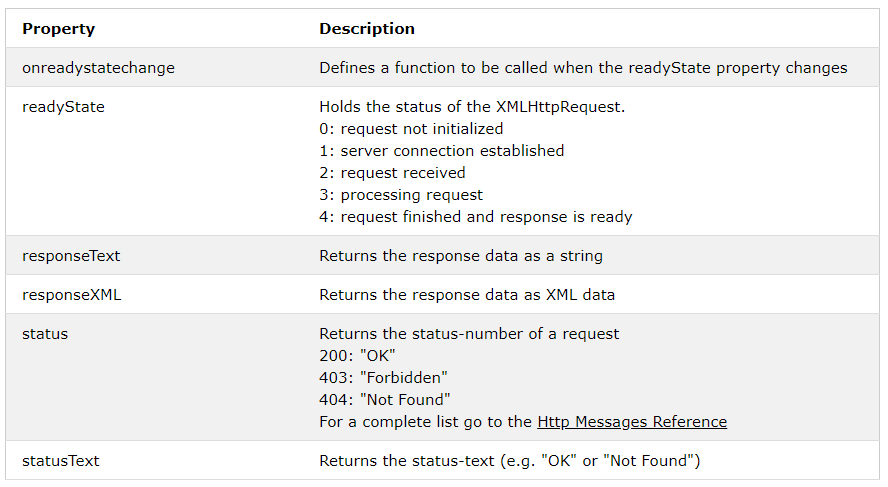
The XMLHttpRequest object can be used to exchange data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

## XMLHttpRequest Object Methods



## 

## XMLHttpRequest Object Properties

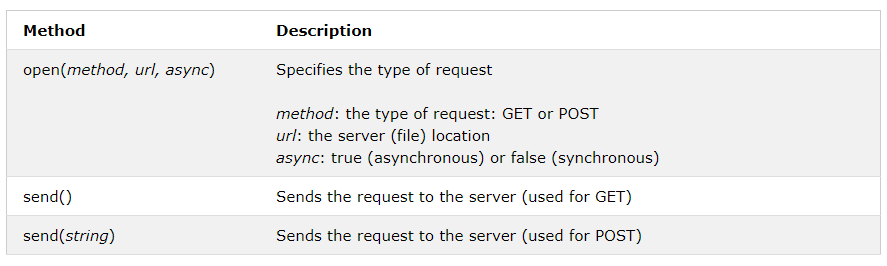


# AJAX - Send a Request To a Server

The XMLHttpRequest object is used to exchange data with a server.

## Send a Request To a Server

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object



## GET or POST?

GET is simpler and faster than POST, and can be used in most cases.

However, always use POST requests when:

1. A cached file is not an option (update a file or database on the server).
2. Sending a large amount of data to the server (POST has no size limitations).
3. Sending user input (which can contain unknown characters), POST is more robust and secure than GET.

## The url - A File On a Server

The url parameter of the open() method, is an address to a file on a server. The file can be any kind of file, like .txt and .xml, or server scripting files like .asp and .php (which can perform actions on the server before sending the response back).

## Asynchronous - True or False?

Server requests should be sent asynchronously.

The async parameter of the open() method should be set to true



By sending asynchronously, the JavaScript does not have to wait for the server response, but can instead:

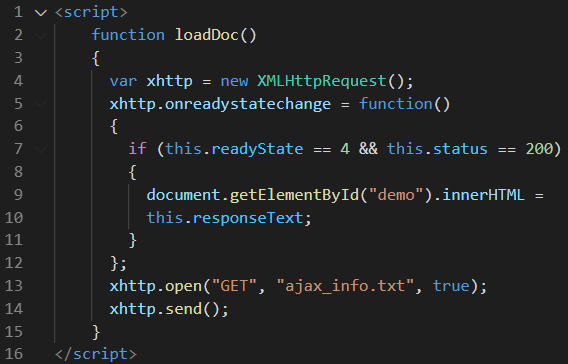
1. execute other scripts while waiting for server response
2. deal with the response after the response is ready

## The onreadystatechange Property

With the XMLHttpRequest object you can define a function to be executed when the request receives an answer.

The function is defined in the onreadystatechange property of the XMLHttpRequest object

Example:



## Synchronous Request

To execute a synchronous request, change the third parameter in the open() method to false:

xhttp.open("GET", "ajax\_info.txt", false);

Sometimes async = false are used for quick testing. You will also find synchronous requests in older JavaScript code.

Since the code will wait for server completion, there is no need for an onreadystatechange function:

Example:



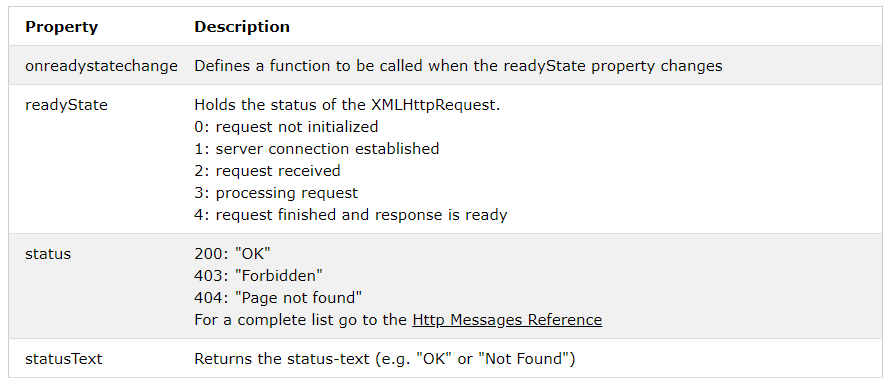
# AJAX - Server Response

## The onreadystatechange Property

The readyState property holds the status of the XMLHttpRequest.

The onreadystatechange property defines a function to be executed when the readyState changes.

The status property and the statusText property holds the status of the XMLHttpRequest object.



The onreadystatechange function is called every time the readyState changes.

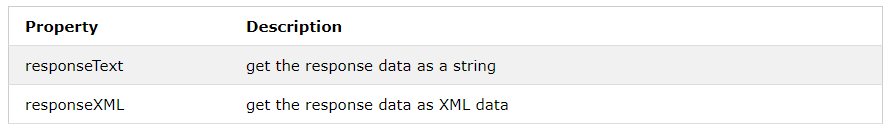
## Using a Callback Function

A callback function is a function passed as a parameter to another function.

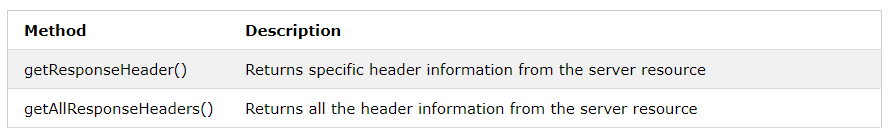
If you have more than one AJAX task in a website, you should create one function for executing the XMLHttpRequest object, and one callback function for each AJAX task.

The function call should contain the URL and what function to call when the response is ready.

## Server Response Properties



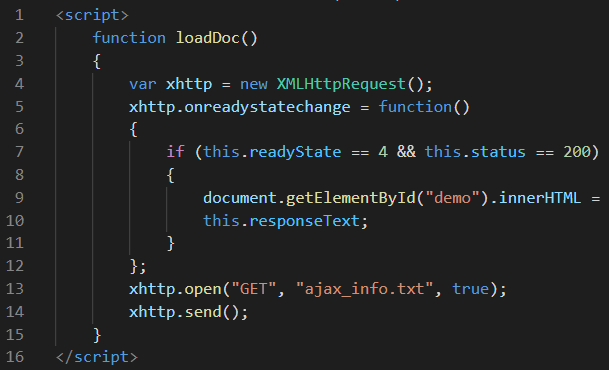
## Server Response Methods



## The responseText Property

The responseText property returns the server response as a JavaScript string, and you can use it accordingly

Example



--------------------------Reference Credits: W3SCHOOLS,Tutorialspoint-----------------------