JavaScript Assignment

**Basic Defination**

1. [What is JavaScript?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript#A_high-level_definition)

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

1. [What is a variable?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Variables#What_is_a_variable)

A variable is a container for a value, like a number we might use in a sum, or a string that we might use as part of a sentence. But one special thing about variables is that their contained values can change.

1. [What are strings?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Strings)

A JavaScript string simply stores a series of characters like "Antra .NET".

A string can be any text inside quotes. You can use single or double quotes:

**Example**

var carname = "Volvo XC60";  
 var carname = 'Volvo XC60';

1. [What is an array?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Arrays#What_is_an_Array)

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

**Example**

var cars = ["Saab", "Volvo", "BMW"];

1. [What is a loop?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Looping_code)

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

Often this is the case when working with arrays:

### Instead of writing:

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

### You can write:

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

1. [What is a function?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Functions)

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

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

**Example:**

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

1. [What is an event?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/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.

1. [What is an object?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics#Object_basics)

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers −

**Encapsulation** − the capability to store related information, whether data or methods, together in an object.

**Aggregation** − the capability to store one object inside another object.

**Inheritance** − the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.

**Polymorphism** − the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

1. [What is JSON?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON#No_really_what_is_JSON)

* JSON stands for **J**ava**S**cript **O**bject **N**otation
* JSON is lightweight data interchange format
* JSON is language independent **\***
* JSON is "self-describing" and easy to understand

The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text only. Code for reading and generating JSON data can be written in any programming language.

1. [What is a Web API?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction#What_are_APIs)

Application Programming Interfaces (APIs) are constructs made available in programming languages to allow developers to create complex functionality more easily. They abstract more complex code away from you, providing some easier syntax to use in its place.

1. [What is the DOM?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Manipulating_documents#The_document_object_model)

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:

* Core DOM - standard model for all document types
* XML DOM - standard model for XML documents
* HTML DOM - standard model for HTML documents

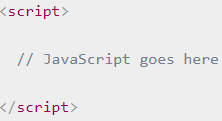
**General**

1. [How do you add JavaScript to your page?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript#How_do_you_add_JavaScript_to_your_page)

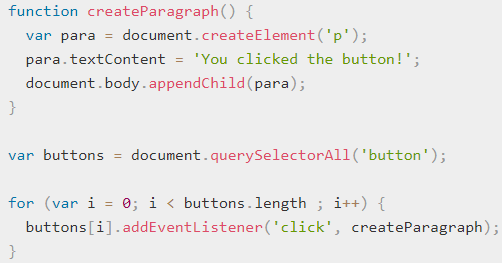
JavaScript is applied to your HTML page in a similar manner to CSS. Whereas CSS uses <link> elements to apply external stylesheets and <style> elements to apply internal stylesheets to HTML, JavaScript only needs one friend in the world of HTML — the <script> element. Let's learn how this works.

**Internal JavaScript**

* First of all, make a local copy of our example file apply-javascript.html. Save it in a directory somewhere sensible.
* Open the file in your web browser and in your text editor. You'll see that the HTML creates a simple web page containing a clickable button.
* Next, go to your text editor and add the following just before your closing </body> tag:



* Now we'll add some JavaScript inside our <script> element to make the page do something more interesting — add the following code just below the "// JavaScript goes here" line:



* Save your file and refresh the browser — now you should see that when you click the button, a new paragraph is generated and placed below.

**External JavaScript**

* First, create a new file in the same directory as your sample HTML file. Call it script.js — make sure it has that .js filename extension, as that's how it is recognized as JavaScript.
* Next, copy all of the script out of your current <script> element and paste it into the .js file. Save that file.
* Now replace your current <script> element with the following:



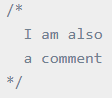
* Save and refresh your browser, and you should see the same thing! It works just the same, but now we've got the JavaScript in an external file. This is generally a good thing in terms of organizing your code, and making it reusable across multiple HTML files. Plus the HTML is easier to read without huge chunks of script dumped in it.

1. [How do you add comments to JavaScript code?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript#Comments)

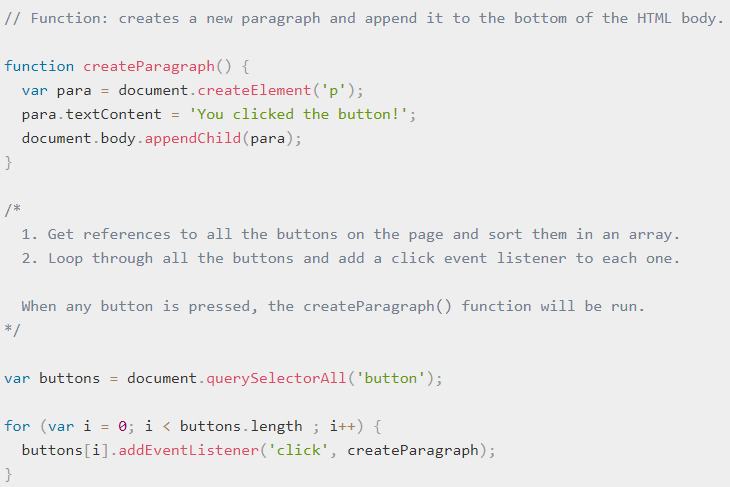
* A single line comment is written after a double forward slash (//), e.g.



* A multi-line comment is written between the strings /\* and \*/, e.g.



So for example, we could annotate our last demo's JavaScript with comments like so:



### Variables

1. [How do you declare a variable?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Variables#Declaring_a_variable)

First got to create a variable. To do this, we type the keyword var followed by the name you want to call your variable:



Here we're creating two variables called myName and myAge.

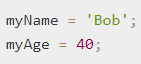
1. [How do you initialize a variable with a value?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Variables#Initializing_a_variable)

Once you've declared a variable, you can initialize it with a value. You do this by typing the variable name, followed by an equals sign (=), followed by the value you want to give it. For example:



1. [How do you update a variable's value?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Variables#Updating_a_variable) (also see [Assignment operators](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Math#Assignment_operators))

Once a variable has been initialized with a value , you can change (or update) that value by simply giving it a different value.



1. [What data types can values have in JavaScript?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Variables#Variable_types)

There are a few different types of data we can store in variables.

**Numbers**

You can store numbers in variables, either whole numbers like 30 (also called integers) or decimal numbers like 2.456 (also called floats or floating point numbers). You don't need to declare variable types in JavaScript, unlike some other programming languages. When you give a variable a number value, you don't include quotes:



**Strings**

Strings are pieces of text. When you give a variable a string value, you need to wrap it in single or double quote marks, otherwise JavaScript will try to interpret it as another variable name.



**Booleans**

Booleans are true/false values — they can have two values, true or false. These are generally used to test a condition, after which code is run as appropriate. So for example, a simple case would be:



Whereas in reality it would be used more like this:



This is using the "less than" operator (<) to test whether 6 is less than 3. As you might expect, it will return false, because 6 is not less than 3! You will learn a lot more about such operators later on in the course.

**Arrays**

An array is a single object that contains multiple values enclosed in square brackets and separated by commas.



Once these arrays are defined, you can access each value by their location within the array.



The square brackets specify an index value corresponding to the position of the value you want returned. You might have noticed that arrays in JavaScript are zero-indexed: the first element is at index 0.

**Objects**

In programming, an object is a structure of code that models a real life object.



To retrieve the information stored in the object, you can use the following syntax:



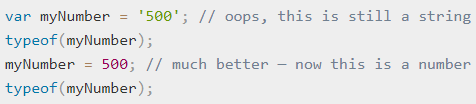
1. [What does 'loosely typed' mean?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Variables#Loose_typing)

JavaScript is a "loosely typed language", which means that, unlike some other languages, you don't need to specify what data type a variable will contain (e.g. numbers, strings, arrays, etc).

For example, if you declare a variable and give it a value encapsulated in quotes, the browser will treat the variable as a string:



It will still be a string, even if it contains numbers, so be careful:



### Math

1. [What types of number do you have to deal with in web development?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Math#Types_of_numbers)

There are different types of decimal numbers, for example:

* **Integers** are whole numbers, e.g. 10, 400, or -5.
* **Floating point numbers** (floats) have decimal points and decimal places, for example 12.5, and 56.7786543.
* **Doubles** are a specific type of floating point number that have greater precision than standard floating point numbers (meaning that they are accurate to a greater number of decimal places).

We even have different types of number system! Decimal is base 10 (meaning it uses 0–9 in each column), but we also have things like:

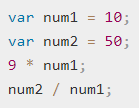
* **Binary** — The lowest level language of computers; 0s and 1s.
* **Octal** — Base 8, uses 0–7 in each column.
* **Hexadecimal** — Base 16, uses 0–9 and then a–f in each column. You may have encountered these numbers before when setting colors in CSS.

1. [How do you do basic math in JavaScript?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Math#Arithmetic_operators)

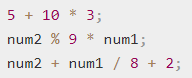
Arithmetic operators are the basic operators that we use to do sums:

| **Operator** | **Name** | **Purpose** | **Example** |
| --- | --- | --- | --- |
| + | Addition | Adds two numbers together. | 6 + 9 |
| - | Subtraction | Subtracts the right number from the left. | 20 - 15 |
| \* | Multiplication | Multiplies two numbers together. | 3 \* 7 |
| / | Division | Divides the left number by the right. | 10 / 5 |
| % | Remainder (sometimes called modulo) | Returns the remainder left over after you've shared the left number out into a number of integer portions equal to the right number. | 8 % 3 (returns 2, as three goes into 8 twice, leaving 2 left over.) |

* You can also try declaring and initializing some numbers inside variables, and try using those in the sums — the variables will behave exactly like the values they hold for the purposes of the sum. For example:



* Last for this section, try entering some more complex expressions, such as:



1. [What is operator precedence, and how is it handled in JavaScript?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Math#Operator_precedence)

Let's look at the last example from above, assuming that num2 holds the value 50 and num1 holds the value 10 (as originally stated above):



As a human being, you may read this as "50 plus 10 equals 60", then "8 plus 2 equals 10", and finally "60 divided by 10 equals 6".

But the browser does "10 divided by 8 equals 1.25", then "50 plus 1.25 plus 2 equals 53.25".

This is because of operator precedence — some operators will be applied before others when calculating the result of a sum (referred to as an expression, in programming). Operator precedence in JavaScript is the same as is taught in math classes in school — Multiply and divide are always done first, then add and subtract (the sum is always evaluated from left to right).

If you want to override operator precedence, you can put parentheses round the parts that you want to be explicitly dealt with first. So to get a result of 6, we could do this:



1. [How do you increment and decrement values in JavaScript?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Math#Increment_and_decrement_operators)

Sometimes you'll want to repeatedly add or subtract one to/from a numeric variable value. This can be conveniently done using the increment (++) and decrement(--) operators. We used ++ in our "Guess the number" game back in our first splash into JavaScript article, when we added 1 to our guessCount variable to keep track of how many guesses the user has left after each turn.



For a start, note that you can't apply these directly to a number, which might seem strange, but we are assigning a variable a new updated value, not operating on the value itself. The following will return an error:



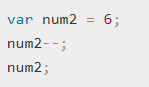
So, you can only increment an existing variable. Try this:



Okay, strangeness number 2! When you do this, you'll see a value of 4 returned — this is because the browser returns the current value, then increments the variable. You can see that it's been incremented if you return the variable value again:



The same is true of -- : try the following



1. [How do you compare values in JavaScript?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Math#Comparison_operators)

We will want to run true/false tests, then act accordingly depending on the result of that test — to do this we use comparison operators.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Purpose** | **Example** |
| === | Strict equality | Tests whether the left and right values are identical to one another | 5 === 2 + 4 |
| !== | Strict-non-equality | Tests whether the left and right values not identical to one another | 5 !== 2 + 3 |
| < | Less than | Tests whether the left value is smaller than the right one. | 10 < 6 |
| > | Greater than | Tests whether the left value is greater than the right one. | 10 > 20 |
| <= | Less than or equal to | Tests whether the left value is smaller than or equal to the right one. | 3 <= 2 |
| >= | Greater than or equal to | Tests whether the left value is greater than or equal to the right one. | 5 >= 4 |

### Strings

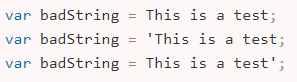
1. [How do you create a string in JavaScript?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Strings#Creating_a_string)

* To start with, enter the following lines:



Just like we did with numbers, we are declaring a variable, initializing it with a string value, and then returning the value. The only difference here is that when writing a string, you need to surround the value with quotes.

* If you don't do this, or miss one of the quotes, you'll get an error. Try entering the following lines:



These lines don't work because any text string without quotes around it is assumed to be a variable name, property name, reserved word, or similar. If the browser can't find it, then an error is raised (e.g. "missing ; before statement"). If the browser can see where a string starts, but can't find the end of the string, as indicated by the 2nd quote, it complains with an error (with "unterminated string literal"). If your program is raising such errors, then go back and check all your strings to make sure you have no missing quote marks.

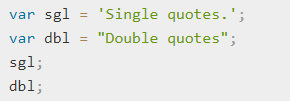
* The following will work if you previously defined the variable string — try it now:



badString is now set to have the same value as string.

1. [Do you have to use single quotes or double quotes?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Strings#Single_quotes_versus_double_quotes)

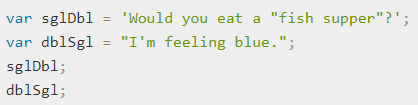
* In JavaScript, you can choose single quotes or double quotes to wrap your strings in. Both of the following will work okay:



* There is very little difference between the two, and which you use is down to personal preference. You should choose one and stick to it, however; differently quoted code can be confusing, especially if you use the different quotes on the same string! The following will return an error:



* The browser will think the string has not been closed, because the other type of quote you are not using to contain your strings can appear in the string. For example, both of these are okay:



* However, you can't include the same quote mark inside the string if it's being used to contain them. The following will error, as it confuses the browser as to where the string ends:



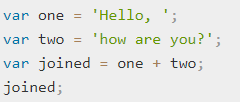
1. [How do you escape characters in strings?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Strings#Escaping_characters_in_a_string)

To fix our previous problem code line, we need to escape the problem quote mark. Escaping characters means that we do something to them to make sure they are recognized as text, not part of the code. In JavaScript, we do this by putting a backslash just before the character.



1. [How do you join strings together?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Strings#Concatenating_strings)

Concatenate is a fancy programming word that means "join together". Joining together strings in JavaScript uses the plus (+) operator, the same one we use to add numbers together, but in this context it does something different.



The result of this is a variable called joined, which contains the value "Hello, how are you?".

In the last instance, we just joined two strings together, but you can do as many as you like, as long as you include a + between each one.

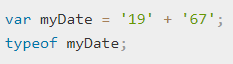


You can also use a mix of variables and actual strings.



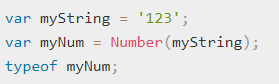
1. [Can you join strings and numbers together?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Strings#Numbers_versus_strings)

You can do this with two numbers — you can force a number to become a string by wrapping it in quote marks. Try the following (we are using the typeof operator to check whether the variable is a number or a string):

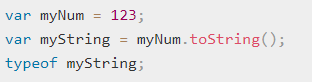


If you have a numeric variable that you want to convert to a string but not change otherwise, or a string variable that you want to convert to a number but not change otherwise, you can use the following two constructs:

* The Number object will convert anything passed to it into a number, if it can.



* On the other hand, every number has a method called toString() that will convert it to the equivalent string.



1. [How do you find the length of a string?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Useful_string_methods#Finding_the_length_of_a_string)

Use the length property



This should return the number 7, because "mozilla" is 7 characters long. This is useful for many reasons; for example, you might want to find the lengths of a series of names so you can display them in order of length, or let a user know that a username they have entered into a form field is too long if it is over a certain length.

1. [How you find what character is at a certain position in a string?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Useful_string_methods#Retrieving_a_specific_string_character)

You can return any character inside a string by using square bracket notation — this means you include square brackets ([]) on the end of your variable name. Inside the square brackets you include the number of the character you want to return, so for example to retrieve the first letter you'd do this:



Computers count from 0, not 1! To retrieve the last character of any string, we could use the following line, combining this technique with the length property we looked at above:



The length of "mozilla" is 7, but because the count starts at 0, the character position is 6, hence us needing length-1. You could use this to, for example, find the first letter of a series of strings and order them alphabetically.

1. [How do you find and extract a specific substring from a string?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Useful_string_methods#Finding_a_substring_inside_a_string_and_extracting_it)

You want to find if a smaller string is present inside a larger. This can be done using the indexOf() method, which takes a single parameter — the substring you want to search for.



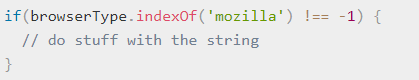
This gives us a result of 2, because the substring "zilla" starts at position 2 (0, 1, 2 — so 3 characters in) inside "mozilla". Such code could be used to filter strings. For example, we may have a list of web addresses and only want to print out the ones that contain "mozilla".

This can be done in another way, which is possibly even more effective.



This should give you a result of -1 — this is returned when the substring, in this case 'vanilla', is not found in the main string.

You could use this to find all instances of strings that don't contain the substring 'mozilla', or do, if you use the negation operator, as shown below. You could do something like this:



When you know where a substring starts inside a string, and you know at which character you want it to end, [slice()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/slice) can be used to extract it.



This returns "moz" — the first parameter is the character position to start extracting at, and the second parameter is the character position after the last one to be extracted. So the slice happens from the first position, up to, but not including, the last position. You could also say that the second parameter is equal to the length of the string being returned.

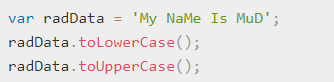
Also, if you know that you want to extract all of the remaining characters in a string after a certain character, you don't have to include the second parameter! Instead, you only need to include the character position from where you want to extract the remaining characters in a string.



This returns "zilla" — this is because the character position of 2 is the letter z, and because you didn't include a second parameter, the substring that was returned was all of the remaining characters in the string.

1. [How do you change the case of a string?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Useful_string_methods#Changing_case)

The string methods toLowerCase() and toUpperCase() take a string and convert all the characters to lower- or uppercase, respectively. This can be useful for example if you want to normalize all user-entered data before storing it in a database.



1. [How do you replace one specific substring with another?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Useful_string_methods#Updating_parts_of_a_string)

You can replace one substring inside a string with another substring using the replace() method. This works very simply at a basic level, although there are some advanced things you can do with it that we won't go into yet.

It takes two parameters — the string you want to replace, and the string you want to replace it with.



Note that to actually get the updated value reflected in the browserType variable in a real program, you'd have to set the variable value to be the result of the operation; it doesn't just update the substring value automatically. So you'd have to actually write this: browserType = browserType.replace('moz','van');

### Arrays

1. [How do you create an array?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Arrays#Creating_an_array)

Arrays are constructed of square brackets, which contain a list of items separated by commas.

* Let's say we wanted to store a shopping list in an array — we'd do something like the following. Enter the following lines into your console:



* In this case, each item in the array is a string, but bear in mind that you can store any item in an array — string, number, object, another variable, even another array. You can also mix and match item types — they don't all have to be numbers, strings, etc.



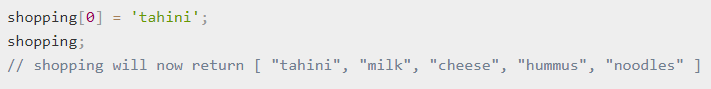
1. [How do you access and modify the items in an array?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Arrays#Accessing_and_modifying_array_items)

You can then access individual items in the array using bracket notation, in the same way that you accessed the letters in a string.

* Enter the following into your console:



* You can also modify an item in an array by simply giving a single array item a new value.



* Note that an array inside an array is called a multidimensional array. You can access an item inside an array that is itself inside another array by chaining two sets of square brackets together. For example, to access one of the items inside the array that is the third item inside the random array (see previous section), we could do something like this:

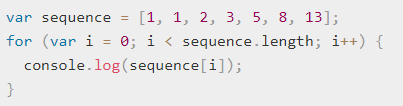


1. [How do you find the length of an array?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Arrays#Finding_the_length_of_an_array)

You can find out the length of an array — by using the length property.



This has other uses, but it is most commonly used to tell a loop to keep going until it has looped through all the items in an array. For example:



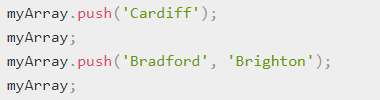
1. [How you add and remove array items?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Arrays#Adding_and_removing_array_items)

First, create the array

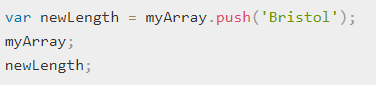


First of all, to add or remove an item at the end of an array we can use push() and pop() respectively.

* + Let's use push() first — note that you need to include one or more items that you want to add to the end of your array.



* + The new length of the array is returned when the method call completes. If you wanted to store the new array length in a variable.



* + Removing the last item from the array is as simple as running pop() on it.



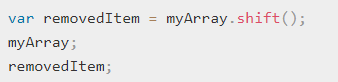
* + The item that was removed is returned when the method call completes.

**unshift()** and **shift()** work in exactly the same way, except that they work on the beginning of the array, not the end.

* + **First unshift() —**

****

* + **shift();—**

****

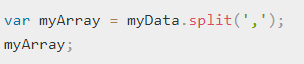
1. [How do you split a string into array items, or join array items into a string?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Arrays#Converting_between_strings_and_arrays)

We can use the split() method. In its simplest form, this takes a single parameter, the character you want to separate the string at, and returns the substrings between the separator as items in an array.

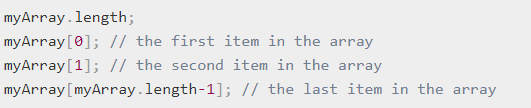
* + First, create a string



* + Now let's split it at each comma:



* + Finally, try finding the length of your new array, and retrieving some items from it:



* + You can also go the opposite way using the join() method.



* + Another way of converting an array to a string is to use the toString() method. toString() is arguably simpler than join() as it doesn't take a parameter, but more limiting. With join() you can specify different separators (try running Step 4 with a different character than a comma).



### Debugging JavaScript

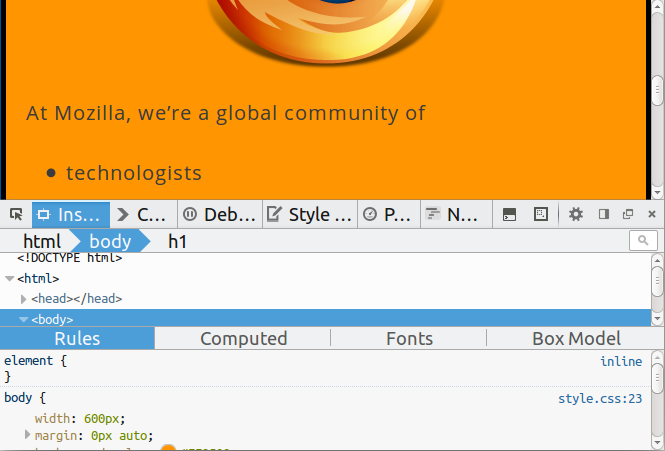
1. [What are the basic types of error?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_went_wrong#Types_of_error)

There are two main types of error there are

* + Syntax errors: These are spelling errors in your code that actually cause the program not to run at all, or stop working part way through — you will usually be provided with some error messages too. These are usually okay to fix, as long as you are familiar with the right tools and know what the error messages mean!
  + Logic errors: These are errors where the syntax is actually correct but the code is not what you intended it to be, meaning that program runs successfully but gives incorrect results. These are often harder to fix than syntax errors, as there usually isn't a resulting error message to direct you to the source of the error.

1. [What are browser developer tools, and how do you access them?](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_are_browser_developer_tools)

The devtools live inside your browser in a subwindow that looks roughly like this:



Three ways:

* **Keyboard.** Ctrl + Shift + I, except

**Internet Explorer.**F12

**Mac OS X.**⌘ + ⌥ + I

* **Menu bar.**

**Firefox.**Menu https://mdn.mozillademos.org/files/9637/2014-01-10-13-08-08-f52b8c.png ➤ ➤ Toggle Tools, or Tools ➤ Web Developer ➤ Toggle Tools

**Chrome.** View ➤ Developer ➤ Developer Tools

**Safari.** Develop ➤ Show Web Inspector. If you can't see the Develop menu, go to Safari ➤ Preferences ➤ Advanced, andcheck the Show Develop menu in menu bar checkbox.

**Opera**. Developer ➤ Web Inspector

* **Context menu.** Press-and-hold/right-click an item on a webpage (Ctrl-click on the Mac), and choose Inspect Element from the context menu that appears. (An added bonus: this method straight-away highlights the code of the element you right-clicked.)

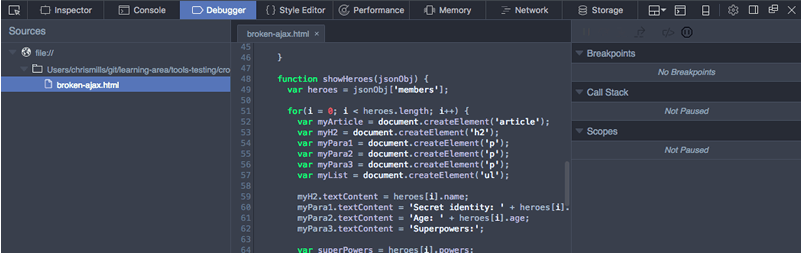
1. [How do you log a value to the JavaScript console?](https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/JavaScript#The_Console_API)

Console API that allows JavaScript code to interact with the browser's JavaScript console. It has a number of features available, but the main one you'll use often is **console.log(),** which prints a custom message to the console.

Anytime something is not working and a value does not appear to be what it is meant to be at some point in your code, you can use **console.log()** to print it out and see what is happening.

1. [How do you use breakpoints, and other JavaScript debugging features?](https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/JavaScript#Using_the_JavaScript_debugger)

In Firefox, the Debugger tab looks as follows:



* + On the left, you can select the script you want to debug (in this case we have only one).
  + The center panel shows the code in the selected script.
  + The right-hand panel shows useful details pertaining to the current environment — Breakpoints, Callstack and currently active Scopes.

The main feature of such tools is the ability to add breakpoints to code — these are points where the execution of the code stops, and at that point you can examine the environment in its current state and see what is going on.

### Making decisions in code

1. [How do you execute different blocks of code, depending on a variable's value or other condition?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/conditionals)

Conditional statements allow us to represent such decision making in JavaScript, from the choice that must be made (e.g. "one cookie or two"), to the resulting outcome or those choices (perhaps the outcome of "ate one cookie" might be "still felt hungry", and the outcome of "ate two cookies" might be "felt full up, but mom scolded me for eating all the cookies".)

The **if statement** executes a statement if a specified condition is true. If the condition is false, another statement can be executed.

Multiple **if...else statements** can be nested to create an else if clause. Note that there is no elseif (in one word) keyword in JavaScript.

A **block statement** (or **compound statement** in other languages) is used to group zero or more statements. The block is delimited by a pair of curly brackets and may optionally be labelled.

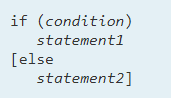
The **switch statement** evaluates an expression, matching the expression's value to a **case** clause, and executes **statements** associated with that case.

The **conditional (ternary) operator** is the only JavaScript operator that takes three operands. This operator is frequently used as a shortcut for the **if** statement.

1. [How do you use if ...else statements?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/conditionals#if_..._else_statements)

The **if statement** executes a statement if a specified condition is truthy. If the condition is falsy, another statement can be executed.

**Syntax**

****

**Condition**

An [expression](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators#Expressions) that is considered to be either truthy or falsy.

**Statement1**

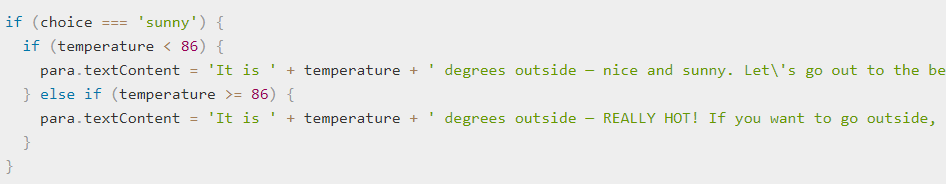
Statement that is executed if condition is truthy. Can be any statement, including further nested if statements. To execute multiple statements, use a [block](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/block) statement ({ ... }) to group those statements. To execute no statements, use an [empty](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/Empty)statement.

**Statement2**

Statement that is executed if condition is falsy and the else clause exists. Can be any statement, including block statements and further nested if statements.

1. [How do nest one decision block inside another?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/conditionals#Nesting_if_..._else)

It is perfectly OK to put one if...else statement inside another one — to nest them. For example, we could update our weather forecast application to show a further set of choices depending on what the temperature is:



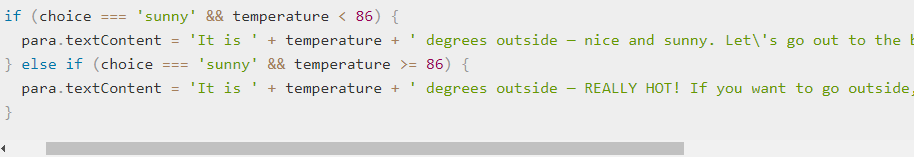
Even though the code all works together, each if...else statement works completely independently of the other one.

1. [How do you use AND, OR, and NOT operators in JavaScript?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/conditionals#Logical_operators_AND_OR_and_NOT)

**&&** — AND; allows you to chain together two or more expressions so that all of them have to individually evaluate to true for the whole expression to return true.

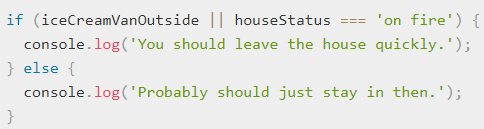
**||** — OR; allows you to chain together two or more expressions so that one or more of them have to individually evaluate to true for the whole expression to return true.

To give you an AND example, the previous example snippet can be rewritten to this:

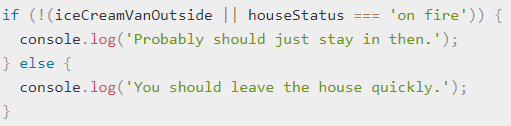


So for example, the first code block will only be run if choice === 'sunny' and temperature < 86 return true.

Let's look at a quick OR example:

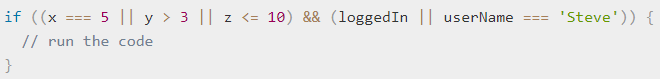


The last type of logical operator, **NOT**, expressed by the **!** operator, can be used to negate an expression. Let's combine it with OR in the above example:



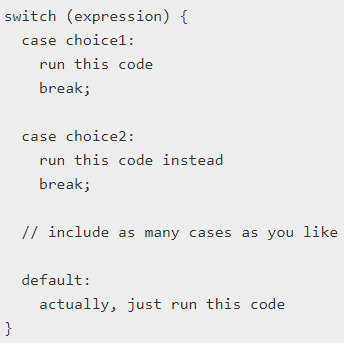
In this snippet, if the OR statement returns true, the NOT operator will negate it so that the overall expression returns false.

You can combine as many logical statements together as you want, in whatever structure. The following example executes the code inside only if both OR statements return true, meaning that the overall AND statement will return true:



1. [How do you conveniently handle a large number of choices for one condition?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/conditionals#switch_statements)

Switch Statements take a single expression/value as an input, and then look through a number of choices until they find one that matches that value, executing the corresponding code that goes along with it.



* The keyword switch, followed by a set of parentheses.
* An expression or value inside the parentheses.
* The keyword case, followed by a choice that the expression/value could be, followed by a colon.
* Some code to run if the choice matches the expression.
* A break statement, followed by a semi-colon. If the previous choice matches the expression/value, the browser stops executing the code block here, and moves on to any code that appears below the switch statement.
* As many other cases (bullets 3–5) as you like.
* The keyword default, followed by exactly the same code pattern as one of the cases (bullets 3–5), except that default does not have a choice after it, and you don't need to break statement as there is nothing to run after this in the block anyway. This is the default option that runs if none of the choices match.

1. [How do you use a ternary operator to make a quick choice between two options based on a true or false test?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/conditionals#Ternary_operator)

The ternary or conditional operator is a small bit of syntax that tests a condition and returns one value/expression if it is true, and another if it is false — this can be useful in some situations, and can take up a lot less code than an if...else block if you simply have two choices that are chosen between via a true/false condition. The pseudocode looks like this:



let's look at a simple example:



Here we have a variable called isBirthday — if this is true, we give our guest a happy birthday message; if not, we give her the standard daily greeting.

**Ternary operator example**

The following live example shows a simple theme chooser where the styling for the site is applied using a ternary operator.

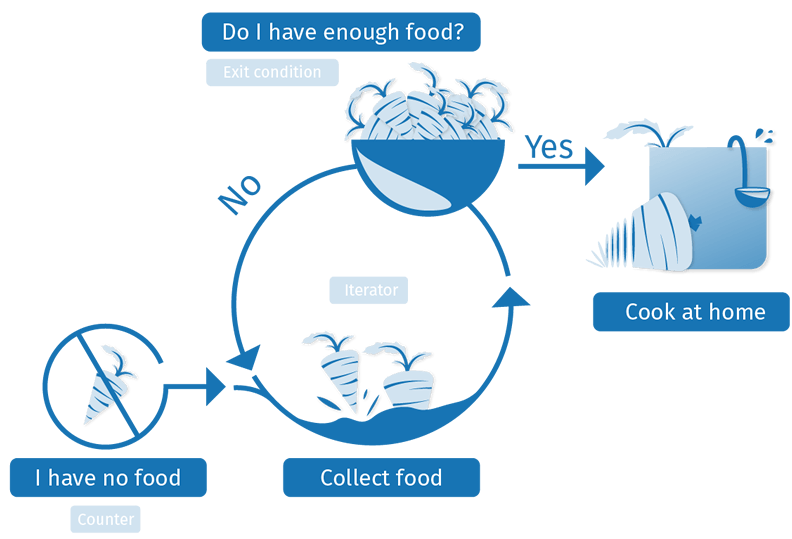


### Looping/iteration

1. [How do you run the same bit of code over and over again?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Looping_code)

Programming loops are all to do with doing the same thing over and over again — which is termed iteration in programming speak.

Let's consider the case of a farmer that is making sure he has enough food to feed his family for the week. He might use the following loop to achieve this:



A loop usually has one or more of the following features:

* A counter, which is initialized with a certain value — this is the starting point of the loop ("Start: I have no food", above).
* An exit condition, which is the criteria under which the loop stops — usually the counter reaching a certain value. This is illustrated by "Have I got enough food?", above. Let's say he needs 10 portions of food to feed his family.
* An iterator, which generally increments the counter by a small amount on each successive loop, until it reaches the exit condition. We haven't explicitly illustrated this above, but we could think about the farmer being able to collect say 2 portions of food per hour. After each hour, the amount of food he has collected is incremented by two, and he checks whether he has enough food. If he has reached 10 portions (the exit condition), he can stop collecting and go home.

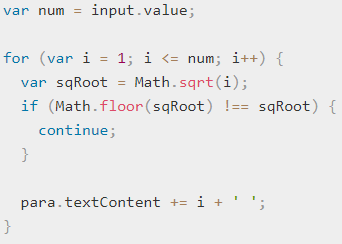
1. [How do you exit a loop before the end if a certain condition is met?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Looping_code#Exiting_loops_with_break)

If you want to exit a loop before all the iterations have been completed, you can use the break statement. When a case is met in a switch statement that matches the input expression, the break statement immediately exits the switch statement and moves onto the code after it.

It's the same with loops — a break statement will immediately exit the loop and make the browser move on to any code that follows it.

1. [How do you skip to the next iteration of a loop if a certain condition is met?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Looping_code#Skipping_iterations_with_continue)

The continue statement works in a similar manner to break, but instead of breaking out of the loop entirely, it skips to the next iteration of the loop. Let's look at another example that takes a number as an input, and returns only the numbers that are squares of integers (whole numbers).



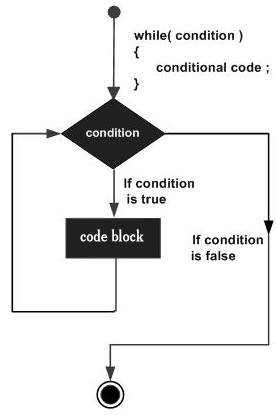
* In this case, the input should be a number (num). The for loop is given a counter starting at 1 (as we are not interested in 0 in this case), an exit condition that says the loop will stop when the counter becomes bigger than the input num, and an iterator that adds 1 to the counter each time.
* Inside the loop, we find the square root of each number using Math.sqrt(i), then check whether the square root is an integer by testing whether it is the same as itself when it has been rounded down to the nearest integer (this is what Math.floor() does to the number it is passed).
* If the square root and the rounded down square root do not equal one another (!==), it means that the square root is not an integer, so we are not interested in it. In such a case, we use the continue statement to skip on to the next loop iteration without recording the number anywhere.
* If the square root IS an integer, we skip past the if block entirely so the continue statement is not executed; instead, we concatenate the current i value plus a space on to the end of the paragraph content.

1. [How do you use while and do ... while loops?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Looping_code#while_and_do_..._while)

The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates.

### Flow Chart

The flow chart of **while loop** looks as follows −



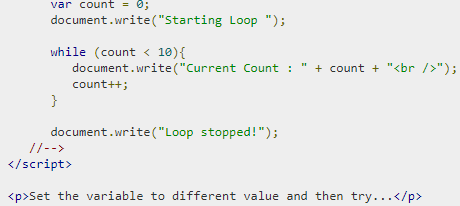
### Syntax

The syntax of **while loop** in JavaScript is as follows −



**Example**

Implement while loop.

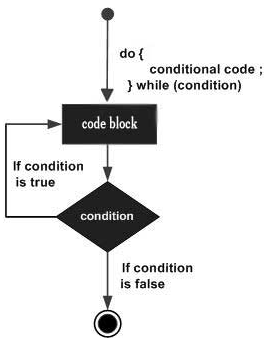
****

**do...while loop**

The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.

**Flow Chart**

The flow chart of a do-while loop would be as follows −



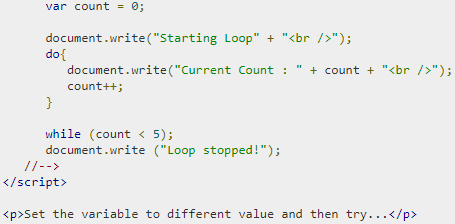
**Syntax**

The syntax for do-while loop in JavaScript is as follows −

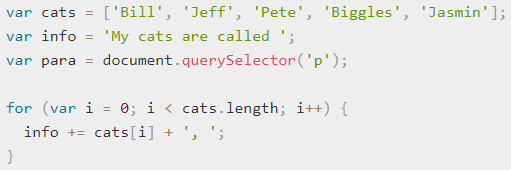


**Example**

Example to how to implement a do-while loop in JavaScript.

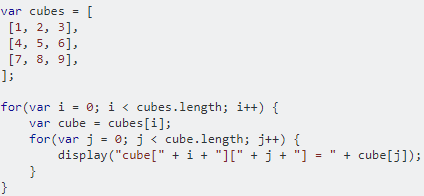


1. How to iterate over the elements in an array

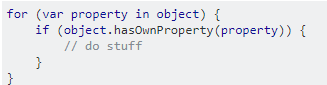
****

* The iterator, i, starts at 0 (var i = 0).
* It has been told to run until it is no longer smaller than the length of the cats array. This is important — the exit condition shows the condition under which the loop will still run. So in this case, while **i < cats.length** is still true, the loop will still run.
* Inside the loop, we concatenate the current loop item (**cats[i]** is cats[whatever **i** is at the time]) along with a comma and a space, onto the end of the info variable. So:
* During the first run**, i = 0**, so **cats[0] + ', '** will be concatenated onto info ("Bill, ").
* During the second run, **i = 1**, so **cats[1] + ', '** will be concatenated onto info ("Jeff, ")
* And so on. After each time the loop has run, 1 will be added to **i (i++),** then the process will start again.
* When **i** becomes equal to **cats.length**, the loop will stop, and the browser will move on to the next bit of code below the loop.

1. How to iterate over the elements in a multidimensional array

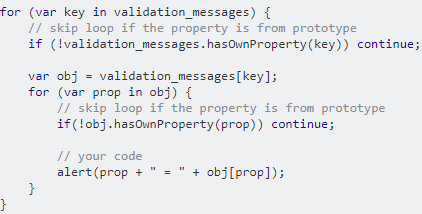


1. How to iterate over the members in an object



hasOwnProperty simply checks to see if this is a property specific to this class, and not one inherited from the base class.

1. How to iterate over the members of an object nested inside an array

****

**Intermediate use cases**

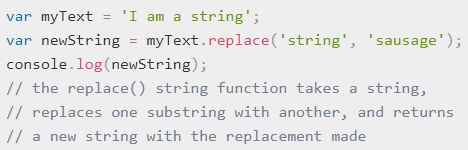
### Functions

1. [How do you find functions in the browser?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Functions#Built-in_browser_functions)

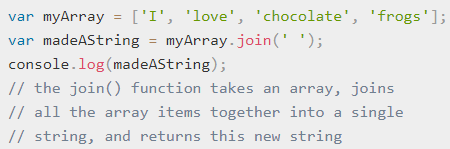
In JavaScript, you'll find functions everywhere. In fact, we've been using functions throughout.

JavaScript structure that features a pair of parentheses — () — and you're not using a common built-in language structure like a for loop, while or do...while loop, or if...else statement, you are making use of a function.

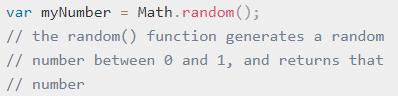
We've made use of functions built in to the browser. Every time we manipulated a text string, for example:



Or every time we manipulated an array:



Or every time we generated a random number:



The JavaScript language has many built-in functions to allows to do useful things without having to write all that code yourself. In fact, some of the code you are calling when you invoke (a fancy word for run, or execute) a built in browser function couldn't be written in JavaScript — many of these functions are calling parts of the background browser code, which is written largely in low-level system languages like C++, not web languages like JavaScript.

Bear in mind that some built-in browser functions are not part of the core JavaScript language — some are defined as part of browser APIs, which build on top of the default language to provide even more functionality

1. [What is the difference between a function and a method?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Functions#Functions_versus_methods)

The distinction is that methods are functions defined inside objects. Built-in browser functions (methods) and variables (which are called properties) are stored inside structured objects, to make the code more efficient and easier to handle.

1. [How do you create your own functions?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Build_your_own_function)

The custom function we are going to build will be called displayMessage(), and it will display a custom message box to the user over the top of a web site. It will function as a more useful replacement for the browser's built-in alert() function.

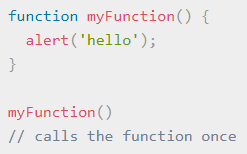


The function takes a single argument — the string that is displayed in the alert box. You can try varying the string to change the message.

The alert function is not really that good: you can alter the message, but you can't easily vary anything else, such as the color, or an icon, or anything else.

1. [How do you run (call, or invoke) a function?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Functions#Invoking_functions)

You've got to run — or invoke — it. This is done by including the name of the function in the code somewhere, followed by parentheses.

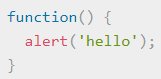


1. [What is an anonymous function?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Functions#Anonymous_functions)

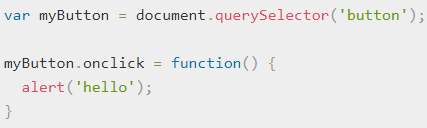
You may see functions defined and invoked in slightly different ways. First we have to create a function:



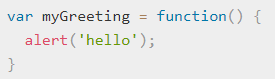
But you can also create a function that doesn't have a name:



This is called an anonymous function — it has no name! It also won't do anything on its own. You generally use an anonymous function along with an event handler, for example the following would run the code inside the function whenever the associated button is clicked:



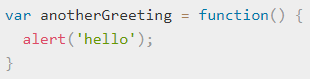
The above example would require there to be a <button> element available on the page to select and click.



This function could now be invoked using:



This effectively gives the variable a name; you can also assign the function to be the value of multiple variables, for example:



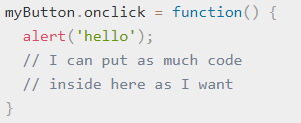
This function could now be invoked using either of



But this would just be confusing, so don't do it! When creating functions, it is better to just stick to this form:



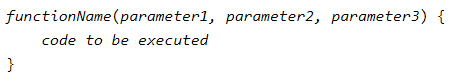
You will mainly use anonymous functions to just run a load of code in response to an event firing — like a button being clicked — using an event handler.



1. [How do you specify parameters (or arguments) when invoking a function?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Functions#Function_parameters)

Function parameters are the names listed in the function definition.

Function arguments are the real values passed to (and received by) the function.



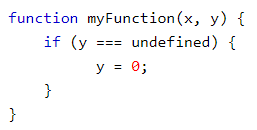
**Parameter Rules**

* JavaScript function definitions do not specify data types for parameters.
* JavaScript functions do not perform type checking on the passed arguments.
* 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:



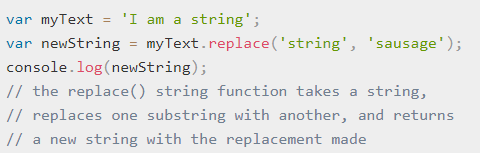
1. [What is function scope?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Functions#Function_scope_and_conflicts)

When you create a function, the variables and other things defined inside the function are inside their own separate scope, meaning that they are locked away in their own separate compartments, unreachable from inside other functions or from code outside the functions.

The top level outside all your functions is called the global scope. Values defined in the global scope are accessible from everywhere in the code.

1. [What are return values, and how do you use them?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Return_values)

Values returned by the function when it completes is called return values.



### Objects

1. [How do you create an object?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics#Object_basics)

An object is a collection of related data and/or functionality (which usually consists of several variables and functions — which are called properties and methods when they are inside objects.)

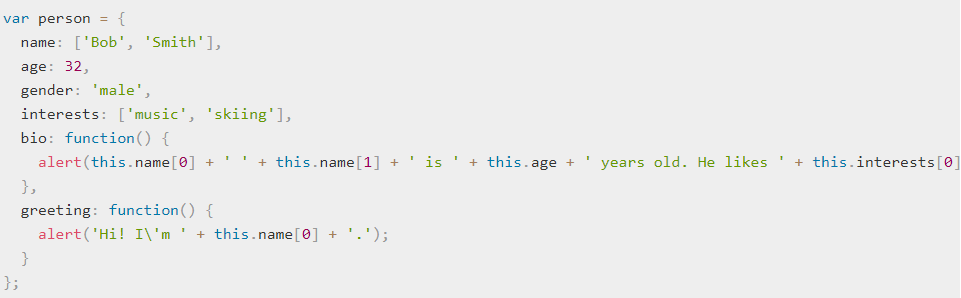
Creating an object often begins with defining and initializing a variable.



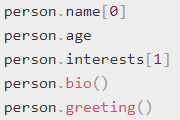
If you enter person into your text input and press the button, you should get the following result:



But this is an empty object, so we can't really do much with it. Let's update our object to look like this:

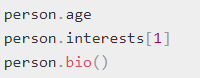


After saving and refreshing, try entering some of the following into your text input:



1. [What is dot notation?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics#Dot_notation)

The object name (person) acts as the namespace — it must be entered first to access anything encapsulated inside the object. Next you write a dot, then the item you want to access — this can be the name of a simple property, an item of an array property, or a call to one of the object's methods, for example:



1. [What is bracket notation?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics#Bracket_notation)

Bracket notation allows access to properties containing special characters and selection of properties using variables.



property\_name is a string. The string does not have to be a valid identifier; it can have any value, e.g. "1foo", "!bar!", or even " " (a space).

This does the exact same thing as the previous example.

A space before bracket notation is allowed.



1. [How do you get and set the methods and properties of an object?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics#Setting_object_members)

You can set (update) the value of object members by simply declaring the member you want to set (using dot or bracket notation), like this:



Try entering these lines, and then getting the members again to see how they've changed:



Setting members doesn't just stop at updating the values of existing properties and methods; you can also create completely new members.



You can now test out your new members:

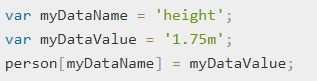
One useful aspect of bracket notation is that it can be used to set not only member values dynamically, but member names too. Let's say we wanted users to be able to store custom value types in their people data, by typing the member name and value into two text inputs? We could get those values like this:



We could then add this new member name and value to the person object like this:



To test this, try adding the following lines into your code, just below the closing curly brace of the person object:

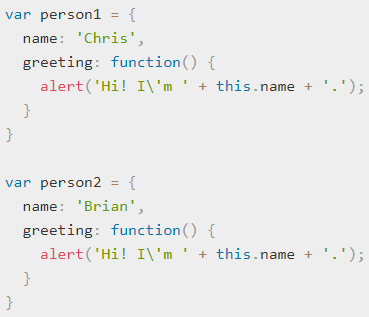


Now try saving and refreshing, and entering the following into your text input:



1. [What is "this", in the context of an object?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics#What_is_this)

The this keyword refers to the current object the code is being written inside — so in this case this is equivalent to person. this is very useful — it will always ensure that the correct values are used when a member's context changes (e.g. two different person object instances may have different names, but will want to use their own name when saying their greeting).



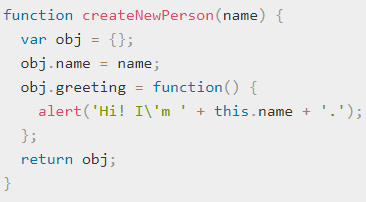
1. [What is object-oriented programming?](https://developer.mozilla.org/docs/Learn/JavaScript/Objects/Object-oriented_JS#Object-oriented_programming_from_10000_meters)

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another.

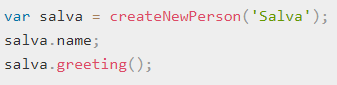
1. [What are constructors and instances, and how do you create them?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_JS#Constructors_and_object_instances)

When a new object instance is created from a constructor function, the functionality is not all copied over to the new object like "classic" OO languages — instead the functionality is linked to via a reference chain called a prototype chain (see Object prototypes). So this is not true instantiation, strictly speaking — JavaScript uses a different mechanism to share functionality between objects.

Let's creating classes via constructors and creating object instances from them in JavaScript.



* You can create a new person by calling this function.



1. [What different ways are there to create objects in JavaScript?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_JS#Other_ways_to_create_object_instances)

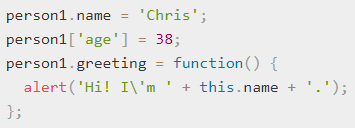
**The Object() constructor**

First of all, you can use the Object() constructor to create a new object. Yes, even generic objects have a constructor, which generates an empty object.

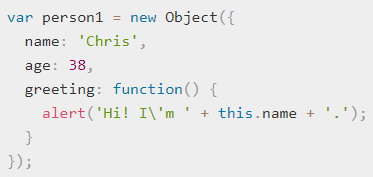
* Try entering this into your browser's JavaScript console:



* This stores an empty object in the person1 variable. You can then add properties and methods to this object using dot or bracket notation as desired.



* You can also pass an object literal to the Object() constructor as a parameter, to prefill it with properties/methods.



**Using the create() method**

Constructors can helps to give your code order— can create constructors in one place, then create instances as needed, and it is clear where they came from.

You can also prefer to create object instances without first creating constructors, especially if they are creating only a few instances of an object. JavaScript has a built-in method called create() that allows to do that. With it, you can create a new object based on any existing object.





You'll see that person2 has been created based on person1—it has the same properties and method available to it.

One limitation of create() is that IE8 does not support it. So constructors may be more effective if you want to support older browsers.

### JSON

1. [How do you structure JSON data, and read it from JavaScript?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON#JSON_structure)

A JSON is a string whose format very much resembles JavaScript object literal format. You can include the same basic data types inside JSON as you can in a standard JavaScript object — strings, numbers, arrays, booleans, and other object literals. This allows you to construct a data hierarchy, like so:

If we loaded this object into a JavaScript program, parsed in a variable called superHeroes for example, we could then access the data inside it using the same dot/bracket notation we looked at in the JavaScript object basics article. **For example:**



To access data further down the hierarchy, you simply have to chain the required property names and array indexes together. For example, to access the third superpower of the second hero listed in the members list, you'd do this:



* First we have the variable name — superHeroes.
* Inside that we want to access the members property, so we use ["members"].
* members contains an array populated by objects. We want to access the second object inside the array, so we use.
* Inside this object, we want to access the powers property, so we use ["powers"].
* Inside the powers property is an array containing the selected hero's superpowers. We want the third one, so we use.

1. [How can you load a JSON file into a page?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON#Loading_our_JSON)

You can load a JSON file into a page using some nifty DOM manipulation to display it.

1. [How do you convert a JSON object to a text string, and back again?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON#Converting_between_objects_and_text)

We set the ([XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest" \o "XMLHttpRequest objects to interact with servers. You can retrieve data from a URL without having to do a full page refresh. This enables a Web page to update just part of a page without disrupting what the user is doing.)) XHR request to convert the JSON response directly into a JavaScript object using:



Sometimes we'll receive a raw JSON string, and we'll need to convert it to an object ourselves. And when we want to send a JavaScript object across the network, we'll need to convert it to JSON (a string) before sending. Luckily, these two problems are so common in web development that a built-in JSON object is available in browsers, which contains the following two methods:

* **parse():** Accepts a JSON string as a parameter, and returns the corresponding JavaScript object.
* **stringify():** Accepts an object as a parameter, and returns the equivalent JSON string form.

### Events

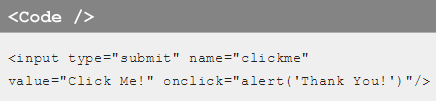
1. [What are event handlers and how do you use them?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events#Event_handler_properties)

To allow you to run your bits of code when these events occur, JavaScript provides us with event handlers. All the event handlers in JavaScript start with the word on, and each event handler deals with a certain type of event. Here's a list of all the event handlers in JavaScript, along with the objects they apply to and the events that trigger them:

**Using an event handler**

To use an event handler, you usually place the event handler name within the HTML tag of the object you want to work with, followed by ="SomeJavaScriptCode", where SomeJavaScriptCode is the JavaScript you would like to execute when the event occurs.

**For Example**

****

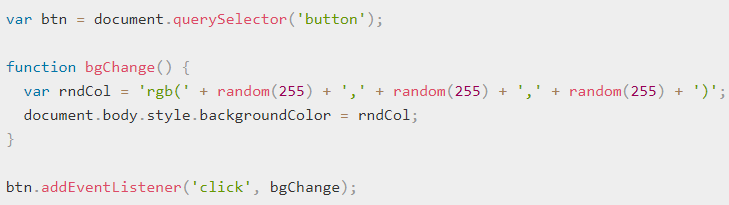
1. [What are inline event handlers?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events#Inline_event_handlers_%E2%80%94_don%27t_use_these)

Inline events are bound to an element by their attribute name, which starts with the "on" prefix. Not all event types may be bound to all elements. For instance, onchange is usually associated with text input fields. Here is a list of some common HTML events:

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

1. [What does the addEventListener() function do, and how do you use it?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events#addEventListener()_and_removeEventListener())

This functions in a similar way to the event handler properties, but the syntax is obviously different. We could rewrite our random color example to look like this:



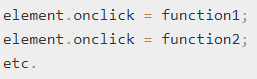
1. [Which mechanism should I use to add event code to my web pages?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events#What_mechanism_should_I_use)

Of the three mechanisms, you definitely shouldn't use the HTML event handler attributes — these are outdated, and bad practice, as mentioned above.

The other two are relatively interchangeable, at least for simple uses:

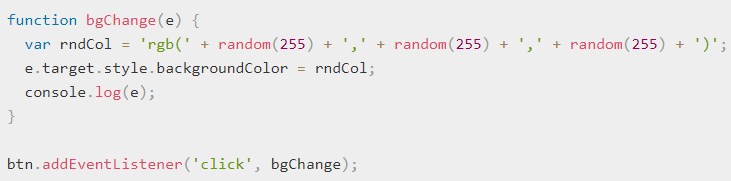
* Event handler properties have less power and options, but better cross browser compatibility (being supported as far back as Internet Explorer 8). You should probably start with these as you are learning.
* DOM Level 2 Events (addEventListener(), etc.) are more powerful, but can also become more complex and are less well supported (supported as far back as Internet Explorer 9). You should also experiment with these, and aim to use them where possible.

The main advantages of the third mechanism are that you can remove event handler code if needed, using removeEventListener(), and you can add multiple listeners of the same type to elements if required. For example, you can call addEventListener('click', function() { ... }) on an element multiple times, with different functions specified in the second argument. This is impossible with event handler properties, because any subsequent attempts to set a property will overwrite earlier ones, e.g.:



1. [What are event objects, and how do you use them?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events#Event_objects)

Sometimes inside an event handler function you might see a parameter specified with a name such as event, evt, or simply e. This is called the event object, and it is automatically passed to event handlers to provide extra features and information. For example, let's rewrite our random color example again slightly:



1. [How do you prevent default event behaviour?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events#Preventing_default_behaviour)

We implement a very simple check inside an onsubmit event handler (the submit event is fired on a form when it is submitted) that tests whether the text fields are empty. If they are, we call the preventDefault() function on the event object — which stops the form submission — and then display an error message in the paragraph below our form to tell the user what's wrong.

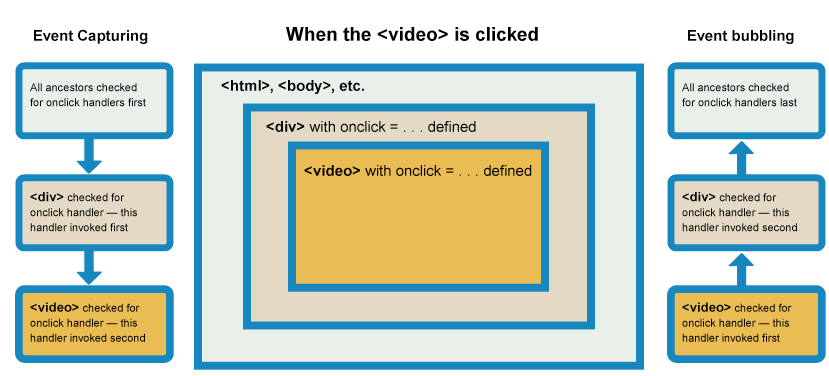
1. [How do events fire on nested elements? (event propagation, also related — event bubbling and capturing)](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events#Event_bubbling_and_capture)

When an event is fired on an element that has parent elements (e.g. the <video> in our case), modern browsers run two different phases — the capturing phase and the bubbling phase. In the capturing phase:

* The browser checks to see if the element's outer-most ancestor (<html>) has an onclick event handler registered on it in the capturing phase, and runs it if so.
* Then it moves on to the next element inside <html> and does the same thing, then the next one, and so on until it reaches the element that was actually clicked on.

In the bubbling phase, the exact opposite occurs:

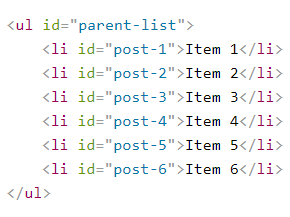
* The browser checks to see if the element that was actually clicked on has an onclick event handler registered on it in the bubbling phase, and runs it if so.
* Then it moves on to the next immediate ancestor element and does the same thing, then the next one, and so on until it reaches the <html> element.



1. [What is event delegation, and how does it work?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events#Event_delegation)

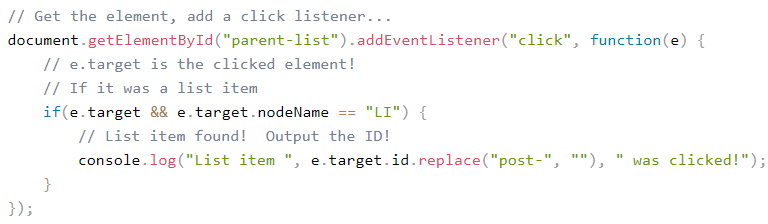
Bubbling also allows us to take advantage of event delegation — this concept relies on the fact that if you want some code to run when you click on any one of a large number of child elements, you can set the event listener on their parent and have the effect of the event listener bubble to each child, rather than having to set the event listener on every child individually.

### Let's say that we have a parent UL element with several child elements:

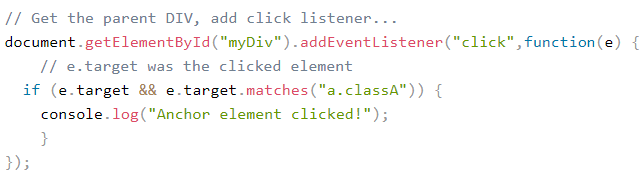


Let's also say that something needs to happen when each child element is clicked. You could add a separate event listener to each individual LI element, but what if LI elements are frequently added and removed from the list? Adding and removing event listeners would be a nightmare, especially if addition and removal code is in different places within your app. The better solution is to add an event listener to the parent UL element. But if you add the event listener to the parent, how will you know which element was clicked?

**Simple:** when the event bubbles up to the UL element, you check the event object's target property to gain a reference to the actual clicked node. Here's a very basic JavaScript snippet which illustrates event delegation:



Start by adding a click event listener to the parent element. When the event listener is triggered, check the event element to ensure it's the type of element to react to. If it is an LI element, boom: we have what we need! If it's not an element that we want, the event can be ignored. This example is pretty simple -- UL and LI is a straight-forward comparison. Let's try something more difficult. Let's have a parent DIV with many children but all we care about is an A tag with the classA CSS class:



### Object-oriented JavaScript

1. [What are object prototypes?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes)

JavaScript is often described as a prototype-based language — each object has a prototype object, which acts as a template object that it inherits methods and properties from. An object's prototype object may also have a prototype object, which it inherits methods and properties from, and so on. This is often referred to as a prototype chain, and explains why different objects have properties and methods defined on other objects available to them.

1. [What is the constructor property, and how can you use it?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes#The_constructor_property)

Constructor Property points to the original constructor function. As you will see in the next section that properties defined on the Person.prototype property (or in general on a constructor function's prototype property, which is an object, as mentioned in the above section) become available to all the instance objects created using the Person() constructor. Hence, the constructor property is also available to both person1 and person2 objects.

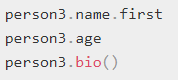
**For example,**

****

These should both return the Person() constructor, as it contains the original definition of these instances.

var person3 = new person1.constructor('Karen', 'Stephenson', 26, 'female', ['playing drums', 'mountain climbing']);

Now try accessing your new object's features, for example:



You won't need to use it often, but it can be really useful when you want to create a new instance and don't have a reference to the original constructor easily available for some reason.

The constructor property has other uses besides. For example, if you have an object instance and you want to return the name of the constructor it is an instance of, you can use the following:

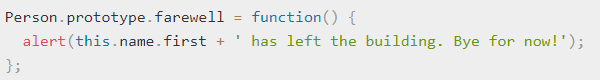


**For example:**

****

1. [How do you add methods to the constructor?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes#Modifying_prototypes)

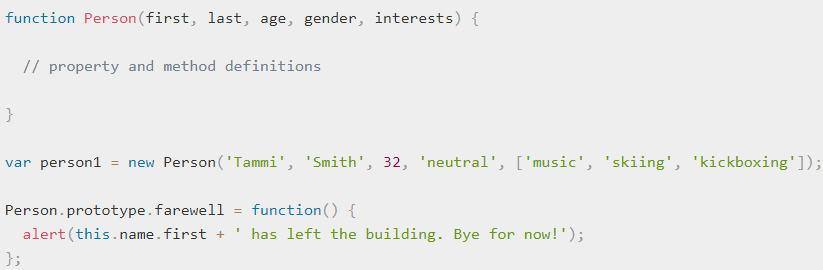
Below the existing JavaScript, add the following code, which adds a new method to the constructor's prototype property:



Save the code and load the page in the browser, and try entering the following into the text input:

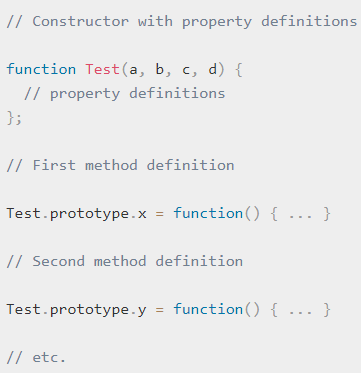


You should get an alert message displayed, featuring the person's name as defined inside the constructor. This is really useful, but what is even more useful is that the whole inheritance chain has updated dynamically, automatically making this new method available on all object instances derived from the constructor.



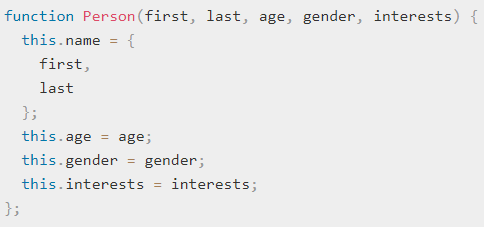
But the farewell() method is still available on the person1 object instance — its available functionality has been automatically updated. This proves what we said earlier about the prototype chain, and the browser looking upwards in the chain to find methods that aren't defined on the object instance itself rather than those methods being copied to the instance.

In fact, a common pattern for more object definitions is to define the properties inside the constructor, and the methods on the prototype. This makes the code easier to read, as the constructor only contains the property definitions, and the methods are split off into separate blocks. For example:

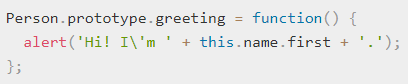


1. [How do you create a new constructor that inherits its members from a parent constructor?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Inheritance)

Person() constructor



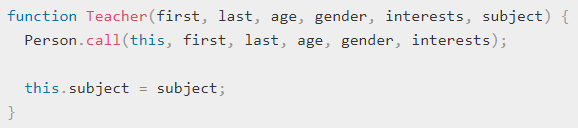
The methods are all defined on the constructor's prototype. For example:



Create a Teacher class, which inherits all the members from Person, but also includes:

* A new property, subject — this will contain the subject the teacher teaches.
* An updated greeting() method, which sounds a bit more formal than the standard greeting() method — more suitable for a teacher addressing some students at school.

Create a Teacher() constructor — add the following below the existing code:



In the above code, call() function is used which allows to call a function defined somewhere else, but in the current context.

Teacher() constructor to take the same parameters as the Person() constructor it is inheriting from, so we specify them all as parameters in the call() invocation.

1. [When should you use inheritance in JavaScript?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Inheritance#Object_member_summary)

In terms of using inheritance in your own code, If you find starting to create a number of objects that have similar features, then creating a generic object type to contain all the shared functionality and inheriting those features in more specialized object types can be convenient and useful.

When using inheritance, you are advised to not have too many levels of inheritance, and to keep careful track of where you define your methods and properties. It is possible to start writing code that temporarily modifies the prototypes of built-in browser objects, but you should not do this unless you have a really good reason. Too much inheritance can lead to endless confusion, and endless pain when you try to debug such code.

### Web APIs

1. [How do you manipulate the DOM (e.g. adding or removing elements) using JavaScript?](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Manipulating_documents#Active_learning_Basic_DOM_manipulation)

To start learning about DOM manipulation, let's begin with a practical example.

* Take a local copy of the dom-example.html page and the image that goes along with it.
* Add a <script></script> element just above the closing </body> tag.
* To manipulate an element inside the DOM, you first need to select it and store a reference to it inside a variable. Inside your script element, add the following line:



Now we have the element reference stored in a variable, we can start to manipulate it using properties and methods available to it (these are defined on intefaces like HTMLAnchorElement in the case of <a> element, its more general parent interface HTMLElement, and Node — which represents all nodes in a DOM). First of all, let's change the text inside the link by updating the value of the Node.textContent property. Add the following line below the previous one:



We should also change the URL the link is pointing to, so that it doesn't go to the wrong place when it is clicked on. Add the following line, again at the bottom:



Note that, as with many things in JavaScript, there are many ways to select an element and store a reference to it in a variable. Document.querySelector() is the recommended modern approach, which is convenient because it allows you to select elements using CSS selectors. The above querySelector() call will match the first <a> element that apprears in the document. If you wanted to match and do things to multiple elements, you could use Document.querySelectorAll(), which matches every element in the document that matches the selector, and stores references to them in an array.

There are older methods available for grabbing element references, such as:

* Document.getElementById(), which selects an element with a given id attribute value, e.g. <p id="myId">My paragraph</p>. The ID is passed to the function as a parameter, i.e. var elementRef = document.getElementById('myId').
* Document.getElementsByTagName(), which returns an array containing all the elements on the page of a given type, for example <p>s, <a>s, etc. The element type is passed to the function as a parameter, i.e. var elementRefArray = document.getElementsByTagName('p').

These two work in older browsers than the modern methods like querySelector(), but are not as convenient. Have a look and see what others you can find!