JavaScript

What is JavaScript

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.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform

History

JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

WHY JAVASCRIPT

- JavaScript is used in millions of Web pages to improve the design, validate forms, detect browsers,and much more.
- JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Mozilla, Firefox, Netscape, Opera.
- API integrations: Developers can use JavaScript to fetch data from other sources and display it on their own site.
- Mobile apps: In the past, you would need other languages to develop great mobile applications, like Objective-C for iOS or Java for Android. It's now easier than before to use JavaScript to connect to mobile APIs though. This means that you can use mobile devices features, such as the camera or localization to build JS-powered apps.

ARE JAVA AND JAVASCRIPT THE SAME?

- NO!
- Java and JavaScript are two completely different languages in both concept and design!
- Java is an object oriented programming language. JavaScript is an object based scripting language.
- Java applications can run in any virtual machine(JVM) or browser. JavaScript code used to run only in browser, but now it can run on server via Node.js.
- Java is a Standalone language. JS contained within a web page and integrates with its HTML content.

Client Side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

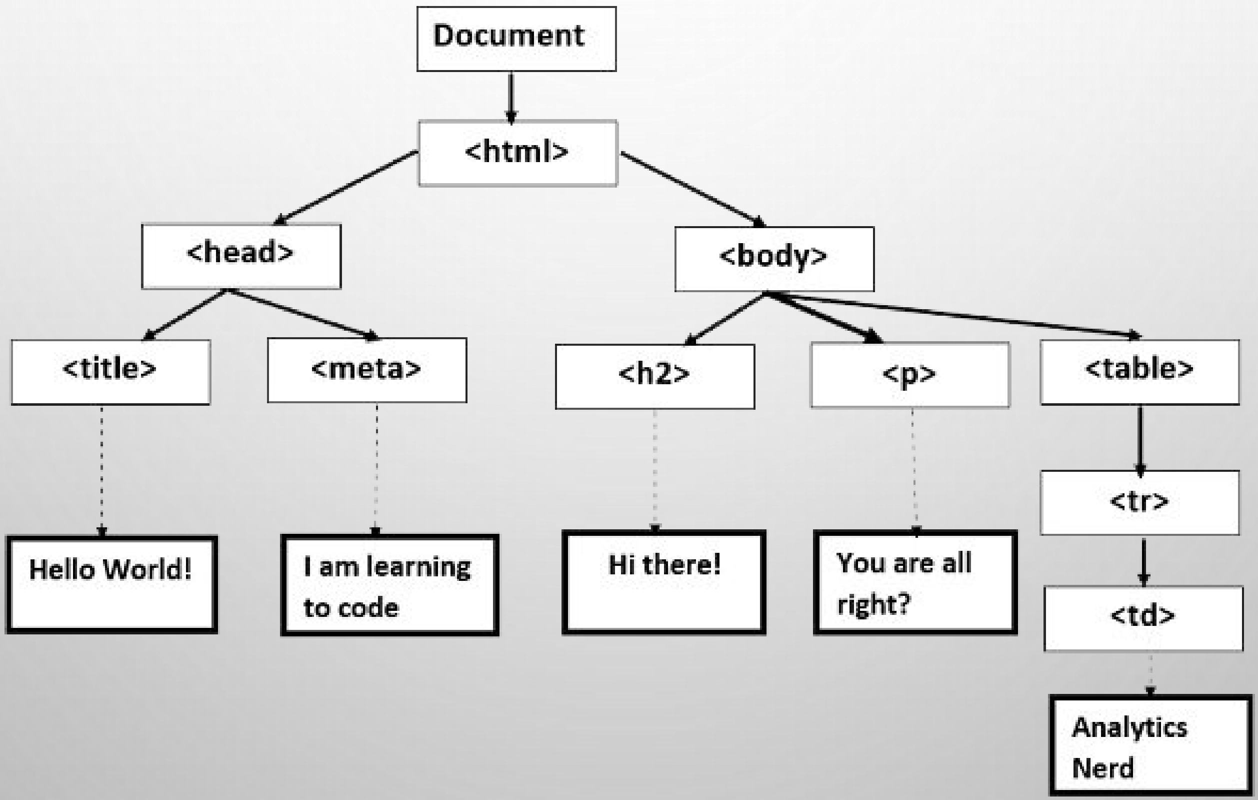
The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. **For example**, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

JAVASCRIPT HTML DOM

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



JAVASCRIPT HTML DOM

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

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

Advantages

- Speed. Client-side JavaScript is very fast because it can be run immediately within the client-side browser. Unless outside resources are required, JavaScript is unhindered by network calls to a backend server.
- Simplicity. JavaScript is relatively simple to learn and implement.
- Popularity. JavaScript is used everywhere on the web.
- Interoperability. JavaScript plays nicely with other languages and can be used in a huge variety of applications.
- Server Load. Being client-side reduces the demand on the website server.
- Gives the ability to create rich interfaces.
- es Advantages

Disadvantages

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multi-threading or multiprocessor capabilities.
- Client-Side Security. Because the code executes on the users' computer, in some cases it can be exploited for malicious purposes. This is one reason some people choose to disable Javascript.
- Browser Support. JavaScript is sometimes interpreted differently by different browsers. This makes it somewhat difficult to write cross-browser code.

JavaScript Nature

It’s dynamic
Many things can be changed. For example, you can freely add and remove properties (fields) of objects after they have been created. And you can directly create objects, without creating an object factory (e.g., a class) first.

It’s dynamically typed
Variables and object properties can always hold values of any type.

It’s functional and object-oriented
JavaScript supports two programming language paradigms: functional programming (first-class functions, closures, partial application via bind(), built-in map() and reduce() for arrays, etc.) and object-oriented programming (mutable state, objects, inheritance, etc.).

It fails silently
JavaScript did not have exception handling until ECMAScript 3. That explains why the language so often fails silently and automatically converts the values of arguments and operands: it initially couldn’t throw exceptions.

JavaScript Nature

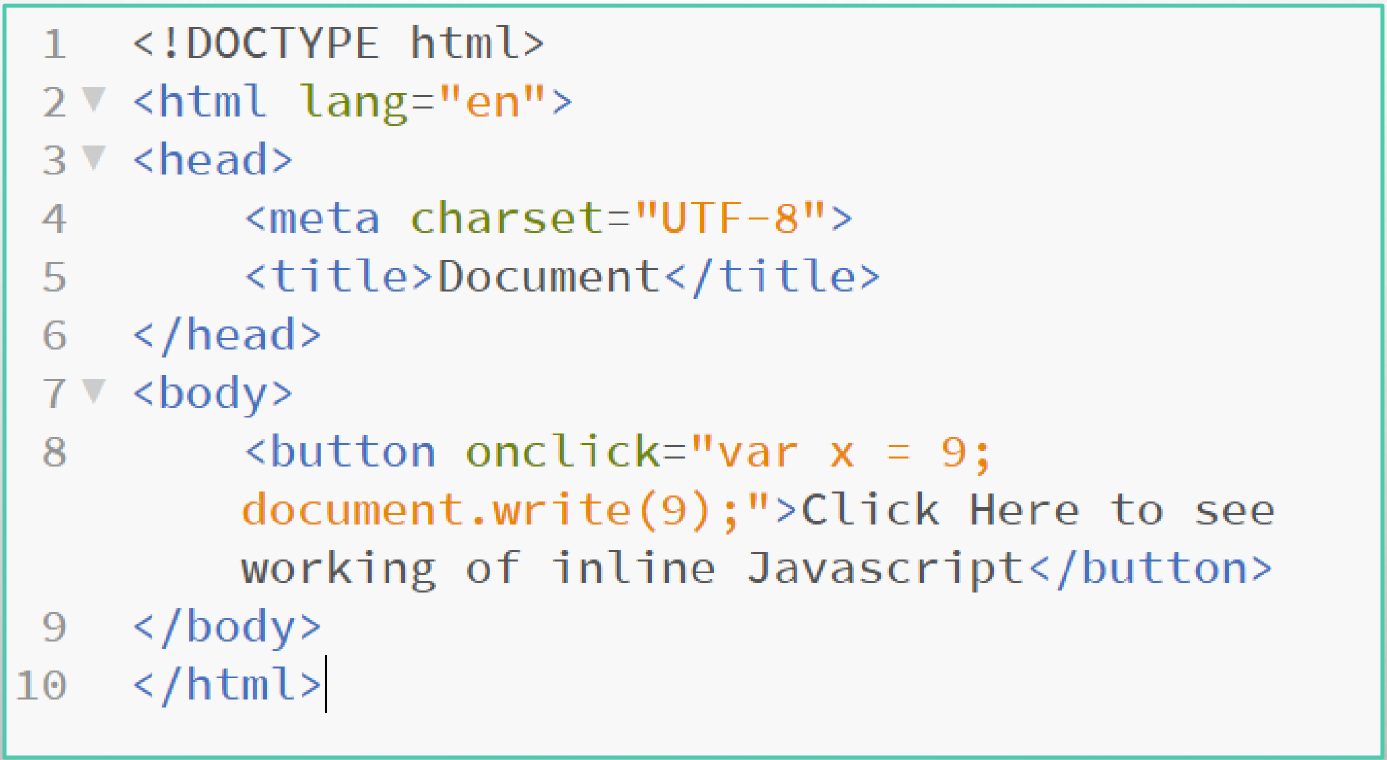
It's deployed as source code
JavaScript is always deployed as source code and compiled by JavaScript engines. Source code has the benefits of being a flexible delivery format and of abstracting the differences between the engines. Two techniques are used to keep file sizes small: compression (mainly gzip) and minification (making source code smaller by renaming variables, removing comments, etc.; see Chapter 32 for details).

It's part of the web platform
JavaScript is such an essential part of the web platform (HTML5 APIs, DOM, etc.) that it is easy to forget that the former can also be used without the latter. However, the more JavaScript is used in nonbrowser settings (such as Node.js), the more obvious it becomes.

HOW TO PUT JAVASCRIPT INTO AN HTML PAGE.

There are following three particular way to put JavaScript into an HTML Page :

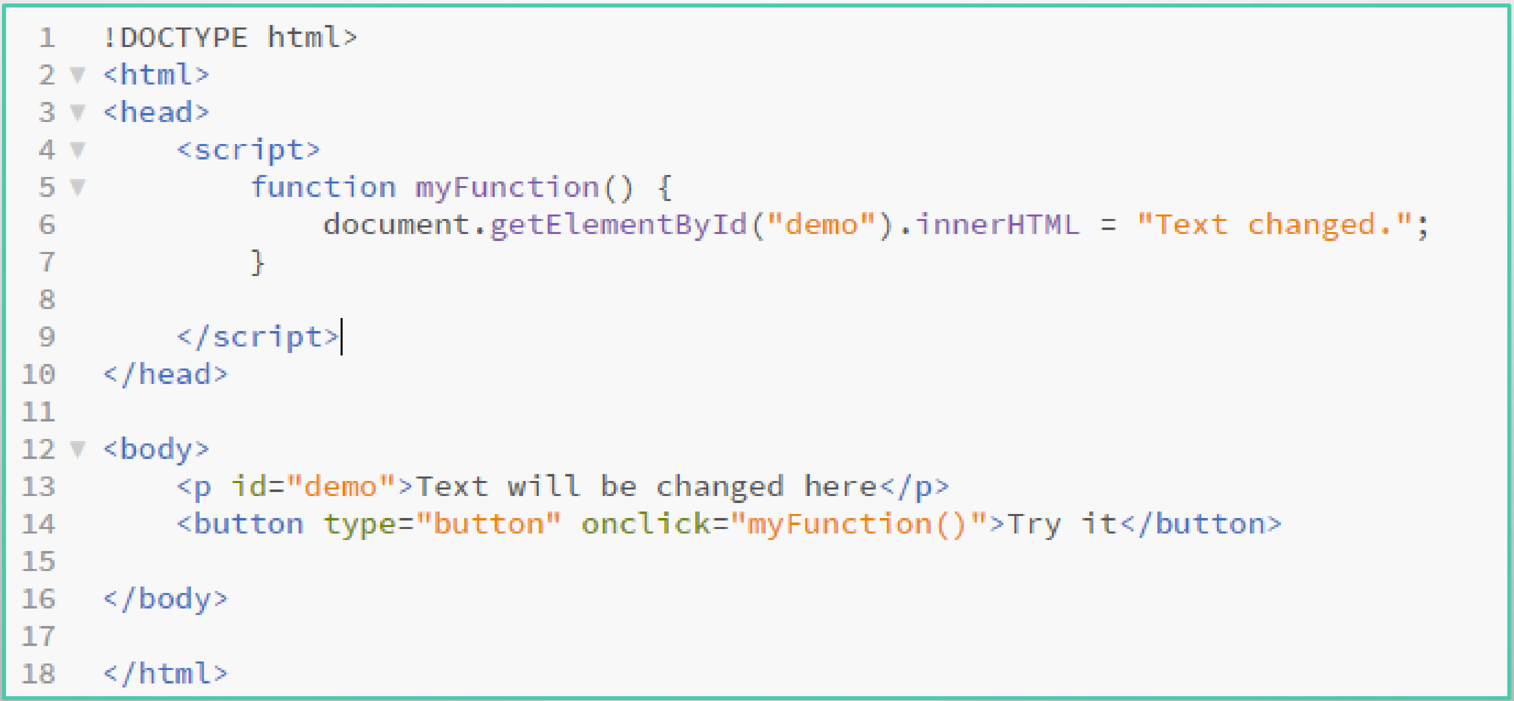
- **Inline JavaScript**: Inline JavaScript is achieved by following methods or we can aslo create a function in script tag and call it by using button tag.



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <button onclick="var x = 9;
    document.write(9);">Click Here to see
    working of inline Javascript</button>
</body>
</html>
```

HOW TO PUT JAVASCRIPT INTO AN HTML PAGE.

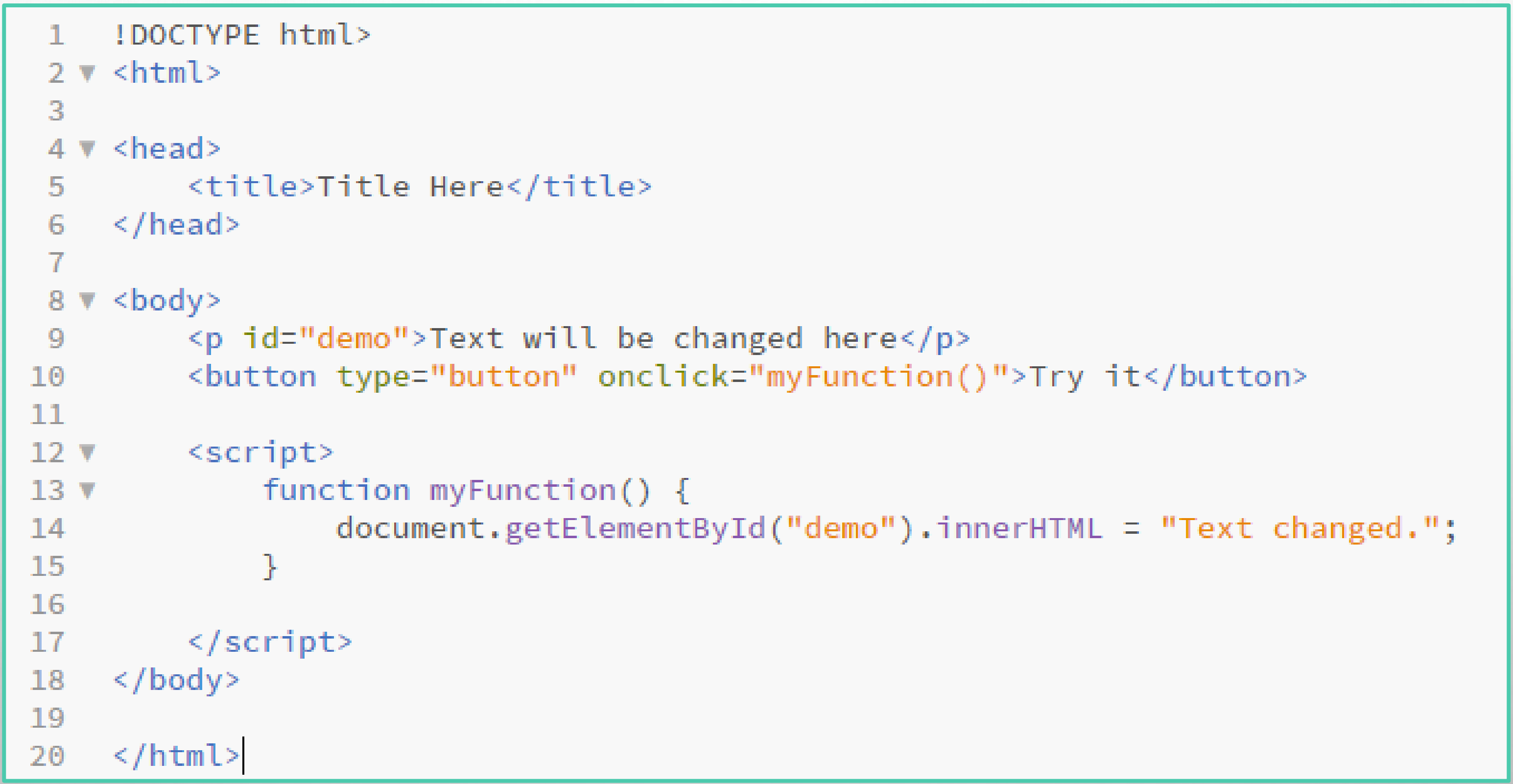
- **Internal JavaScript:**In HTML, JavaScript code is inserted between <script> and </script> tags. You can place any number of scripts in an HTML document. Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.
    - **<u>Script tag in head tag:</u>**



```
!DOCTYPE html>
<html>
<head>
    <script>
        function myFunction() {
            document.getElementById("demo").innerHTML = "Text changed.";
        }

    </script>
</head>

<body>
    <p id="demo">Text will be changed here</p>
    <button type="button" onclick="myFunction()">Try it</button>

</body>

</html>
```

HOW TO PUT JAVASCRIPT INTO AN HTML PAGE.

- **Script tag body tag:**



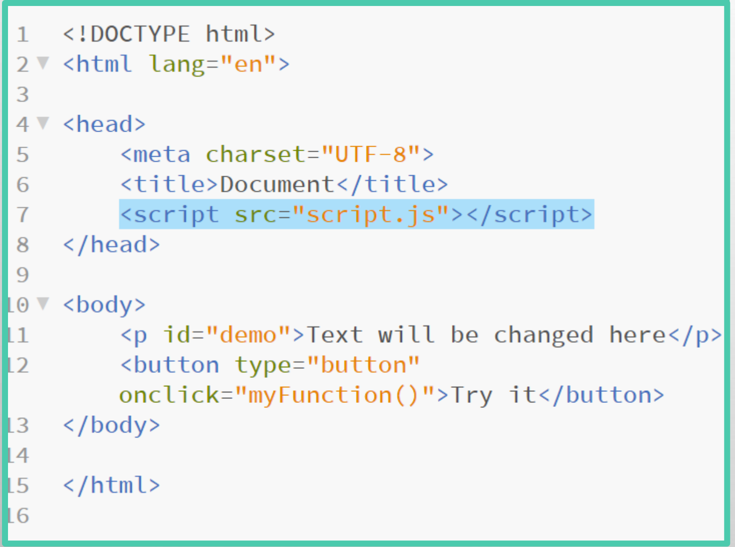
```
!DOCTYPE html>
<html>

<head>
    <title>Title Here</title>
</head>

<body>
    <p id="demo">Text will be changed here</p>
    <button type="button" onclick="myFunction()">Try it</button>

    <script>
        function myFunction() {
            document.getElementById("demo").innerHTML = "Text changed.";
        }

    </script>
</body>

</html>
```

HOW TO PUT JAVASCRIPT INTO AN HTML PAGE.

There are following two particular way to put JavaScript into an HTML Page :

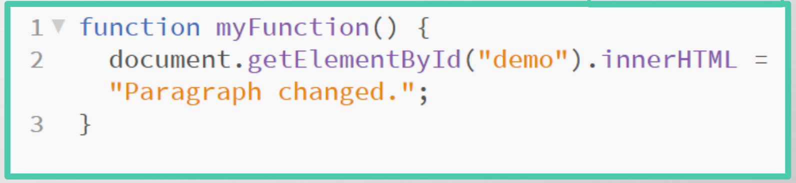
- **External JavaScript:** External scripts are practical when the same code is used in many different web pages. JavaScript files have the file extension **.js** . To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

Index.html



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

<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="script.js"></script>
</head>

<body>
    <p id="demo">Text will be changed here</p>
    <button type="button"
    onclick="myFunction()">Try it</button>
</body>

</html>

```

Script.js



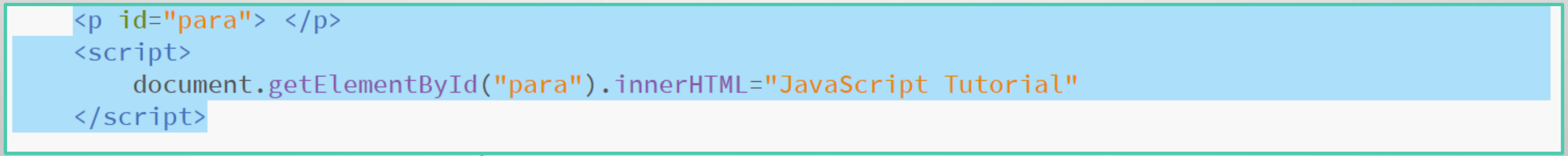
```
function myFunction() {
  document.getElementById("demo").innerHTML =
  "Paragraph changed.";
}
```

JAVASCRIPT OUTPUTS

JavaScript can "display" data in different ways:

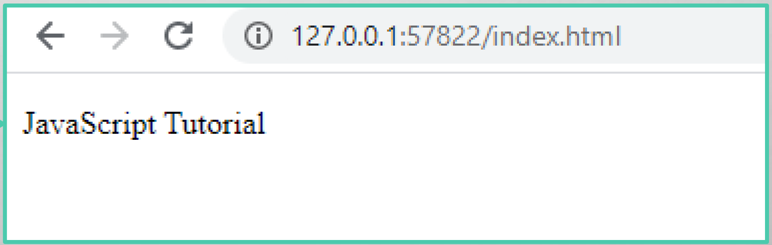
- Writing into an HTML element, using innerHTML.
- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console, using console.log().

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



```
<p id="para"> </p>
<script>
    document.getElementById("para").innerHTML="JavaScript Tutorial"
</script>
```

Result:

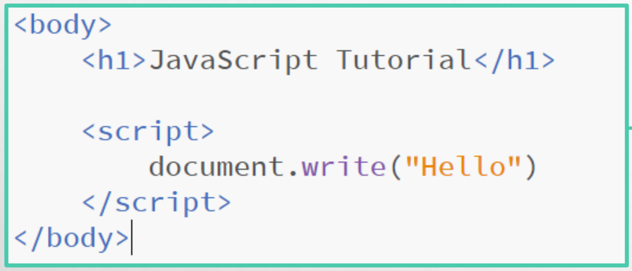


127.0.0.1:57822/index.html

JavaScript Tutorial

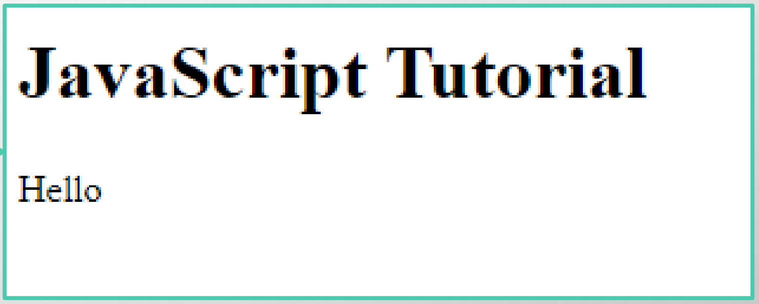
JAVASCRIPT OUTPUTS

- **Using document.write():**It will shoe the mentioned content. Using document.write() after an HTML document is loaded, will delete all existing HTML:



```
<body>
    <h1>JavaScript Tutorial</h1>

    <script>
        document.write("Hello")
    </script>
</body>
```

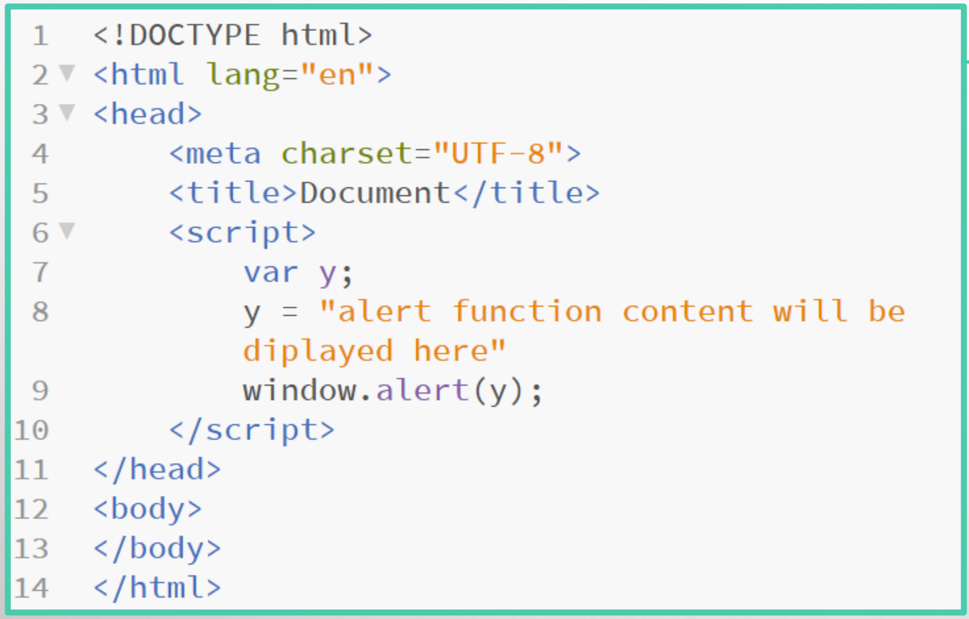
Result:



```
JavaScript Tutorial

Hello
```

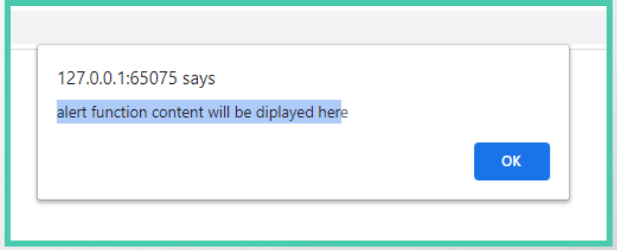
JAVASCRIPT OUTPUTS

using window.alert().



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        var y;
        y = "alert function content will be
        diplayed here"
        window.alert(y);
    </script>
</head>
<body>
</body>
</html>
```

Result:



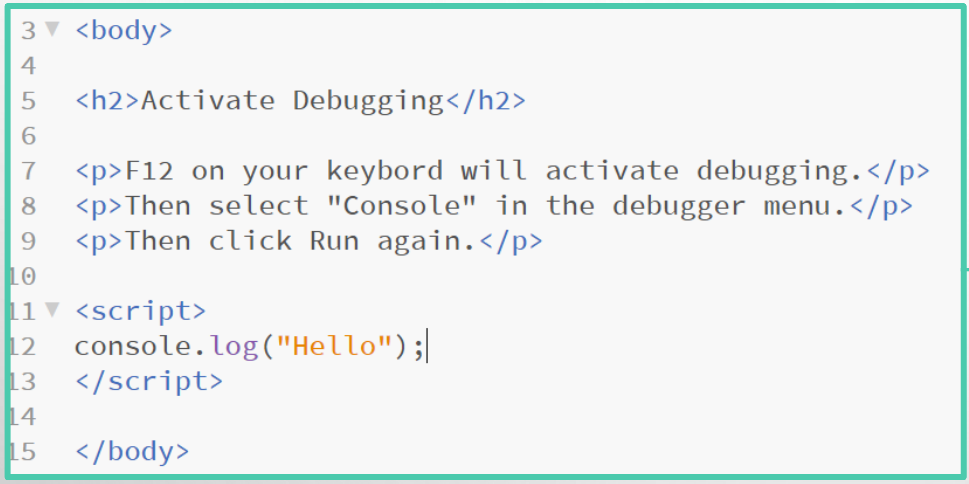
```
127.0.0.1:65075 says
alert function content will be diplayed here
```

An alert box will be appeared at the top center of our browser showing us the output.

JAVASCRIPT OUTPUTS

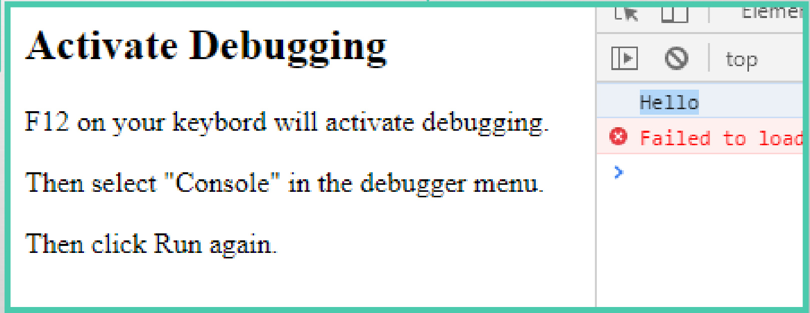
using console.log().

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



```
<body>

<h2>Activate Debugging</h2>

<p>F12 on your keybord will activate debugging.</p>
<p>Then select "Console" in the debugger menu.</p>
<p>Then click Run again.</p>

<script>
console.log("Hello");
</script>

</body>
```

Result:



Activate Debugging

F12 on your keybord will activate debugging.

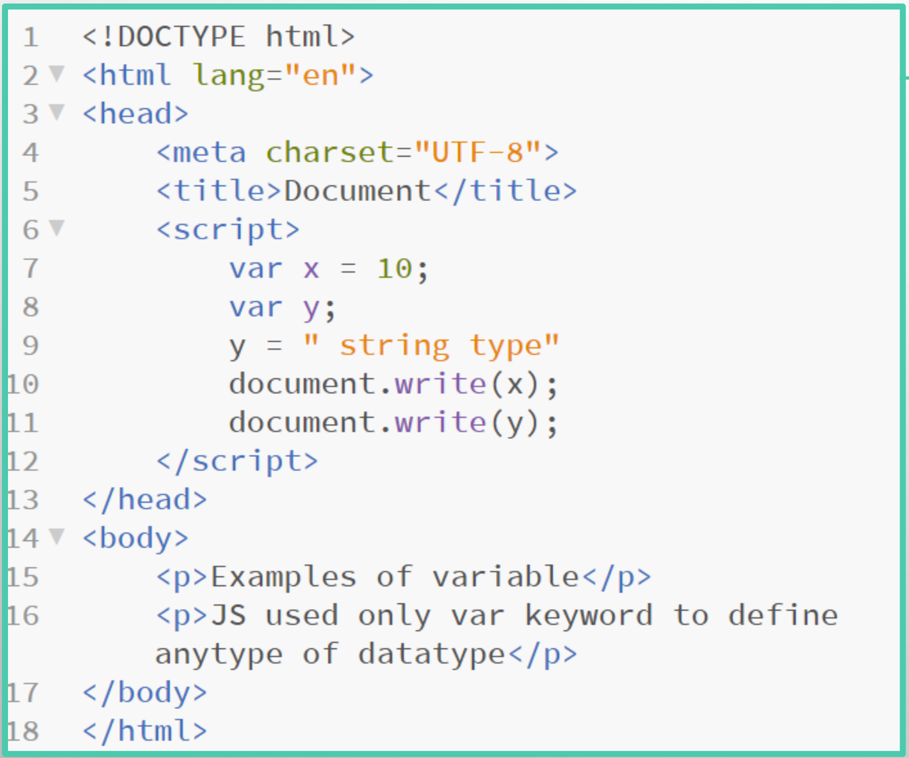
Then select "Console" in the debugger menu.

Then click Run again.

```
Hello
Failed to load
>
```

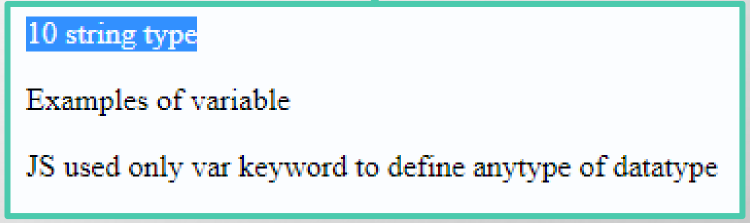
JAVASCRIPT SYNTAX: VARIABLES

- **JavaScript Variables:** In a programming language, **variables** are used to **store** data values. JavaScript uses the **var** keyword to **declare** variables. An **equal sign** is used to **assign values** to variables.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        var x = 10;
        var y;
        y = " string type"
        document.write(x);
        document.write(y);
    </script>
</head>
<body>
    <p>Examples of variable</p>
    <p>JS used only var keyword to define
    anytype of datatype</p>
</body>
</html>
```

Result:



```
10 string type

Examples of variable

JS used only var keyword to define anytype of datatype
```

JavaScript Datatypes

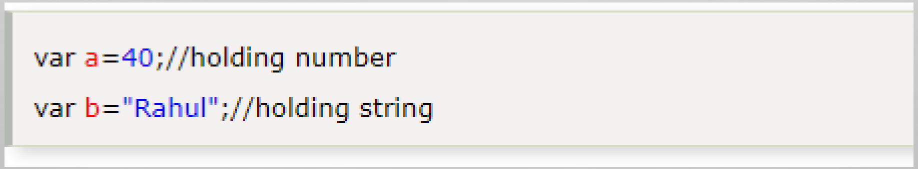
JavaScript Datatypes

JavaScript provides different **data types** to hold different types of values.

There are two types of data types in JavaScript.

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

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



```
var a=40;//holding number
var b="Rahul";//holding string
```

JavaScript Datatypes

<u>JavaScript Primitive Data types</u>

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

JavaScript Datatypes

JavaScript Non-Primitive Data types

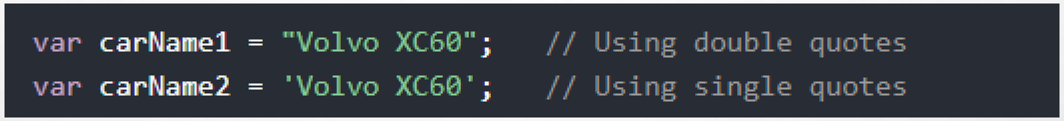
Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

JavaScript Primitives Datatypes

JavaScript Strings:

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

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

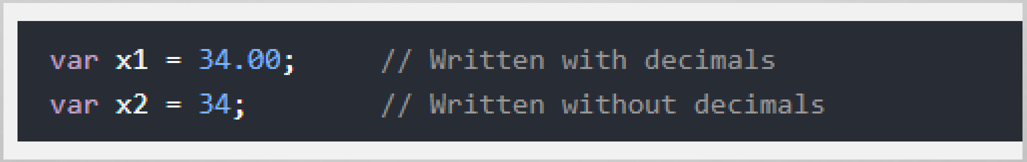


```
var carName1 = "Volvo XC60";   // Using double quotes
var carName2 = 'Volvo XC60';   // Using single quotes
```

JavaScript Numbers

JavaScript has only one type of numbers.

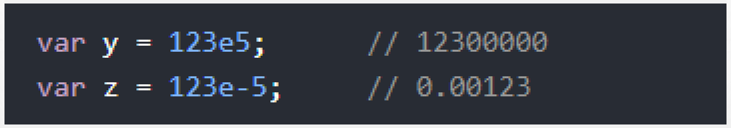
Numbers can be written with, or without decimals:



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

JavaScript Primitives Datatypes

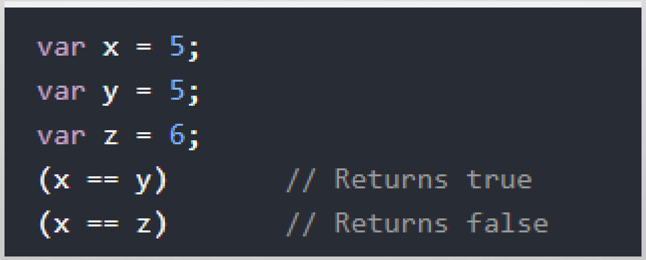
Extra large or extra small numbers can be written with scientific (exponential) notation:



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

JavaScript Booleans

Booleans can only have two values: true or false.



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

JavaScript Primitives Datatypes

Null Variable:

You can assign null to a variable to denote that currently that variable does not have any value but it will have later on. A null means absence of a value.

```
var myVar = null;

alert(myVar); // null
```

Undefined Variable

Undefined is also a primitive value in JavaScript. A variable or an object has an undefined value when no value is assigned before using it. So you can say that undefined means lack of value or unknown value.

```
var myVar;

alert(myVar); // undefined
```

JavaScript Primitives Datatypes

Null Variable:

You can assign null to a variable to denote that currently that variable does not have any value but it will have later on. A null means absence of a value.

```
var myVar = null;

alert(myVar); // null
```

Undefined Variable

Undefined is also a primitive value in JavaScript. A variable or an object has an undefined value when no value is assigned before using it. So you can say that undefined means lack of value or unknown value.

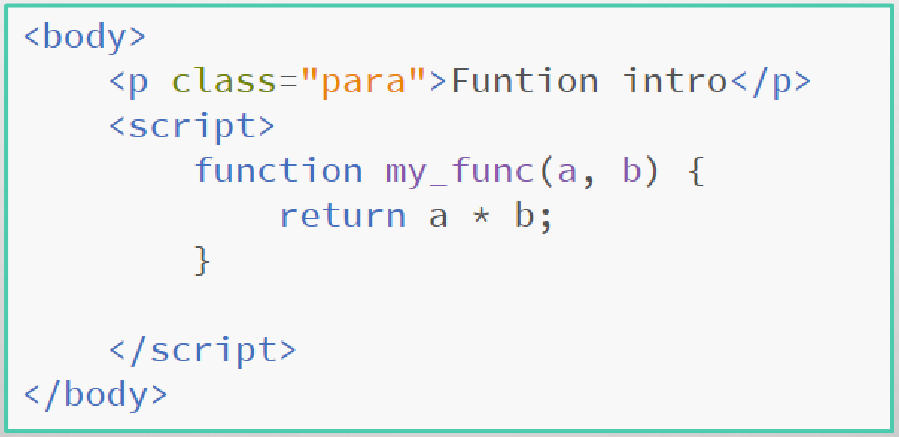
```
var myVar;

alert(myVar); // undefined
```

JAVASCRIPT 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).

Why we need functions:

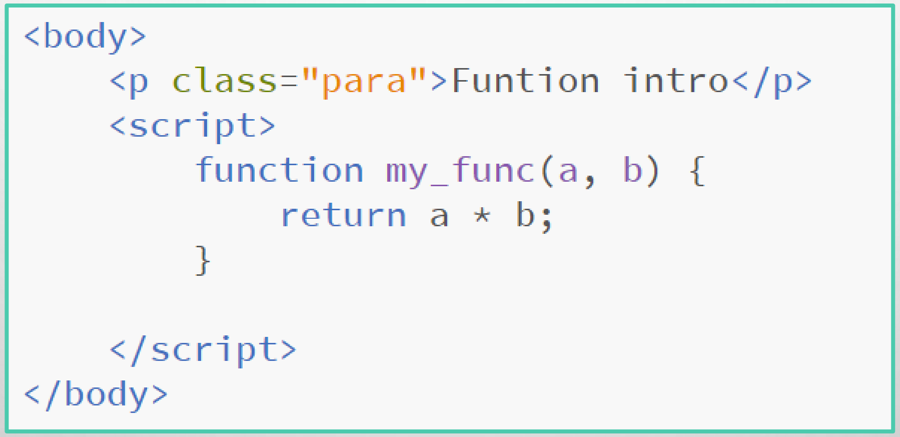
You can reuse code: Define the code once, and use it many times. You can use the same code many times with different arguments, to produce different results.



```
<body>
    <p class="para">Funtion intro</p>
    <script>
        function my_func(a, b) {
            return a * b;
        }

    </script>
</body>
```

JAVASCRIPT FUNCTIONS

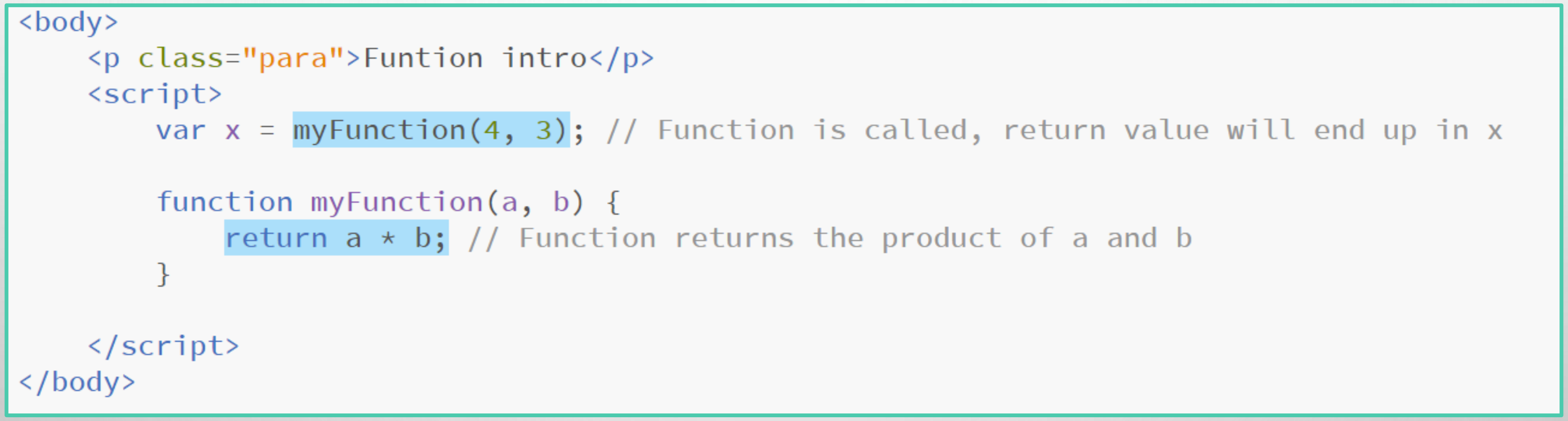
Function Syntax: A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**.Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).The parentheses may include parameter names separated by commas:**(*parameter1*, *parameter2*, ...)**The code to be executed, by the function, is placed inside curly brackets: **{}**



```
<body>
    <p class="para">Funtion intro</p>
    <script>
        function my_func(a, b) {
            return a * b;
        }

    </script>
</body>
```

JAVASCRIPT FUNCTIONS

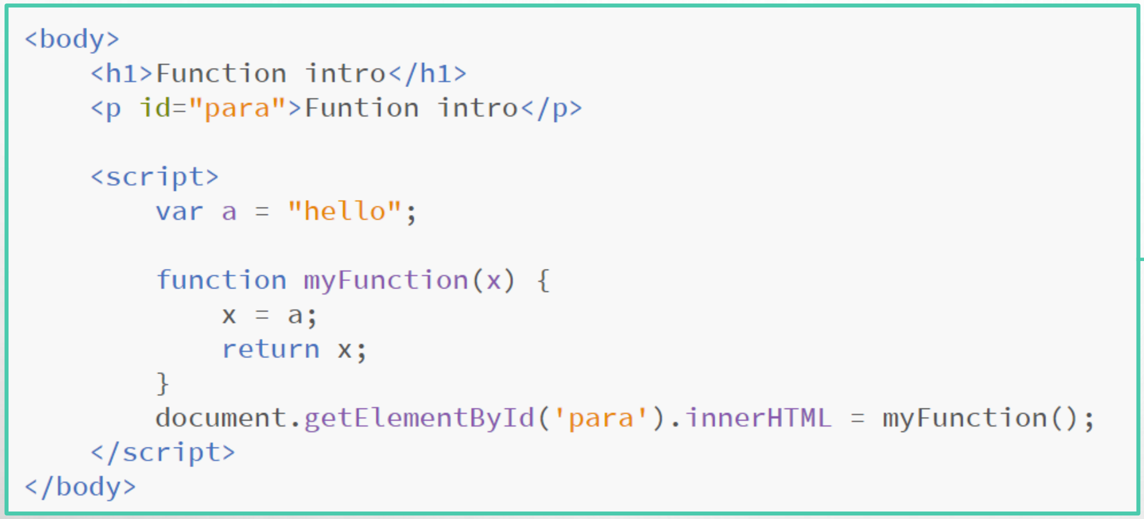
Function Return: When JavaScript reaches a return statement, the function will stop executing. If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement. Functions often compute a **return value**. The return value is "returned" back to the "caller":



```
<body>
    <p class="para">Funtion intro</p>
    <script>
        var x = myFunction(4, 3); // Function is called, return value will end up in x

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

    </script>
</body>
```

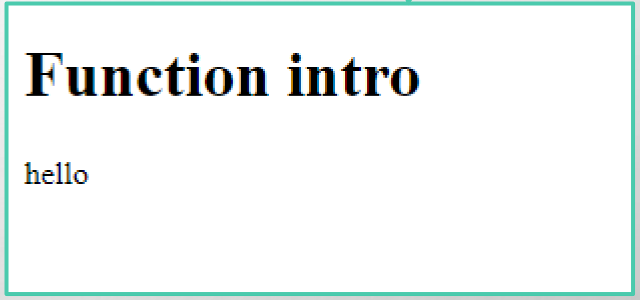
JAVASCRIPT FUNCTIONS

How we call a Function :



```
<body>
    <h1>Function intro</h1>
    <p id="para">Funtion intro</p>

    <script>
        var a = "hello";

        function myFunction(x) {
            x = a;
            return x;
        }
        document.getElementById('para').innerHTML = myFunction();
    </script>
</body>
```

Result:



```
Function intro

hello
```

myFunction refers to the function object, and myFunction() refers to the function result. Accessing a function without () will return the function object instead of the function result.

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.

There are huge number of events in JavaScript. Some few of them are mentioned below:

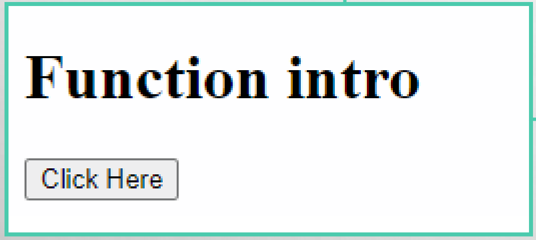
Event	**Description**
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading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

EVENTS:

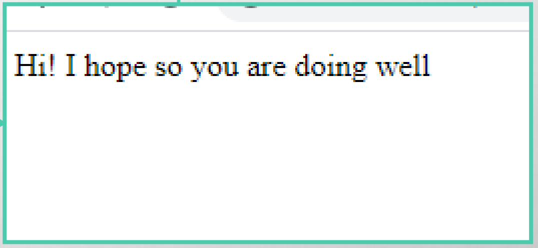
Onclick Event in between the tag:

```
<body>
    <h1>Function intro</h1>
    <p id="para"></p>

    <button onclick="document.write('Hi! I hope so you are doing well')">Click Here</button>

</body></html>
```

Result:



After Button Click:

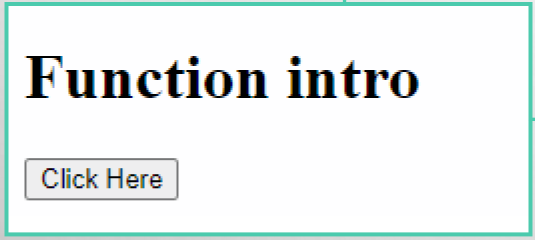

Hi! I hope so you are doing well

EVENTS:

Onclick Event using function call:

```
<button onclick="onclick_demo()">Click Here</button>

<script>
function onclick_demo(){
    document.write('Hi! I hope so you are doing well')
}
</script>
```

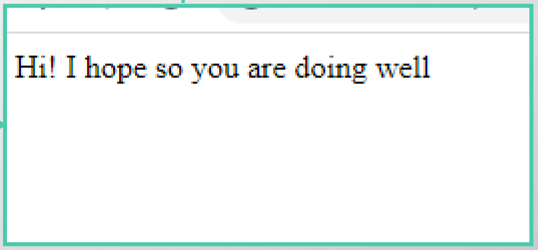
Result:



Function intro

Click Here

After Button Click:



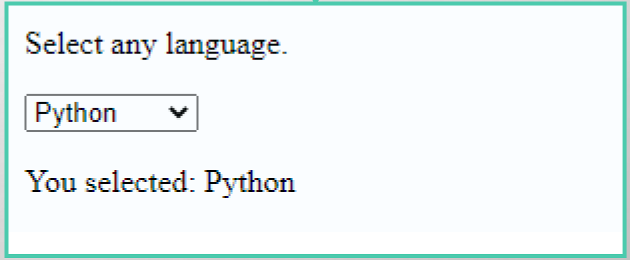
Hi! I hope so you are doing well

EVENTS:

Onchange Event:

```html
<body>
    <p>Select any language.</p>

    <select id="Language" onchange="my_Function()">
        <option value="C++">C++</option>
        <option value="C#">C#</option>
        <option value="Python">Python</option>
        <option value="JavaScript">JavaScript</option>
    </select>
    <p id="demo"></p>

    <script>
        function my_Function() {
            var x = document.getElementById("Language").value;
            document.getElementById("demo").innerHTML = "You selected: " + x;
        }

    </script>
</body>
```

Result:



```
Select any language.

Python

You selected: Python
```

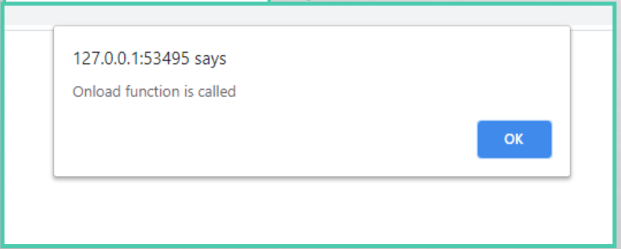
EVENTS:

Onload and onkeydown on Event:

```html
<body onload="my_Function2()">
    <p>A function is triggered when the user is pressing a key in the input field.</p>

    <input type="text" onkeydown="my_Function()">

    <script>
        function my_Function() {
            alert("Any key has just pressed");
        }
        function my_Function2() {
            alert("Onload function is called");
        }
    </script>
</body>
```

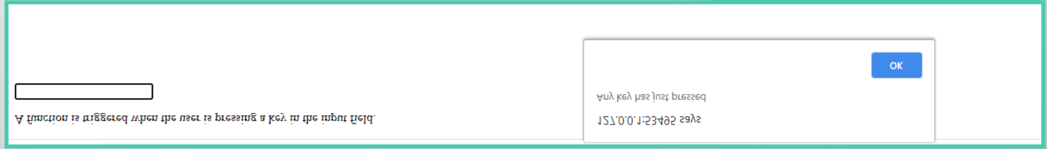
Result:

Onload Function:



```
127.0.0.1:53495 says
Onload function is called
OK
```

Onkeydown Function:



```
A function is triggered when the user is pressing a key in the input field.

127.0.0.1:53495 says
Any key has just pressed
OK
```

JAVASCRIPT OBJECTS

<u>First let we see an Example:</u> <u>Real Life Object, Properties and Methods</u>

In real life, a car is an object. A car has properties like weight and color, and methods like start and stop:

Object	Properties	Methods

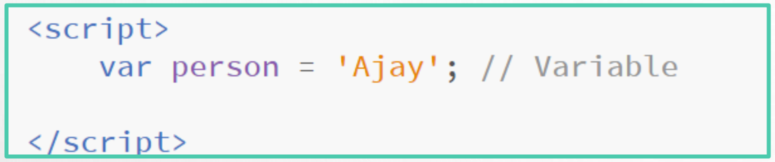
	car.name = Fiat
	car.model = 500
	car.weight = 850kg
	car.color = white
		car.start()
		car.drive()
		car.brake()
		car.stop()

All cars have the same properties, but the property values differ from car to car. All cars have the same methods, but the methods are performed at different times.

JAVASCRIPT OBJECTS

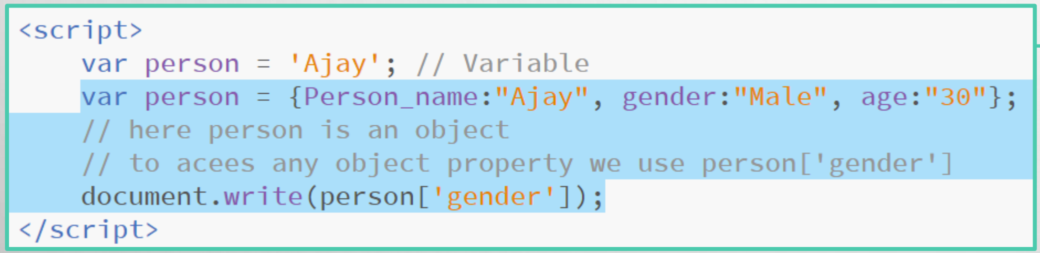
JavaScript Objects:

JavaScript variables are containers for data values. This code assigns a simple value to a variable named person:



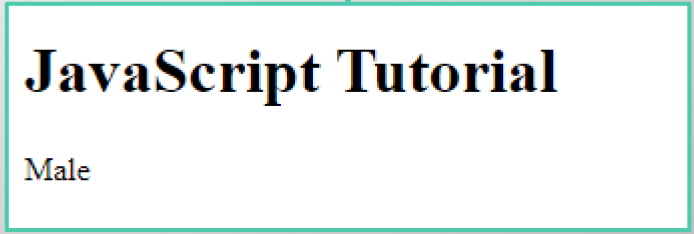
```
<script>
    var person = 'Ajay'; // Variable

</script>
```

Objects are variables too. But objects can contain many values.
This code assigns many values to a variable named person:



```
<script>
    var person = 'Ajay'; // Variable
    var person = {Person_name:"Ajay", gender:"Male", age:"30"};
    // here person is an object
    // to acees any object property we use person['gender']
    document.write(person['gender']);
</script>
```

Result:



JavaScript Tutorial

Male

JAVASCRIPT ARRAYS

Array

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

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

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

Using the JavaScript Keyword new

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

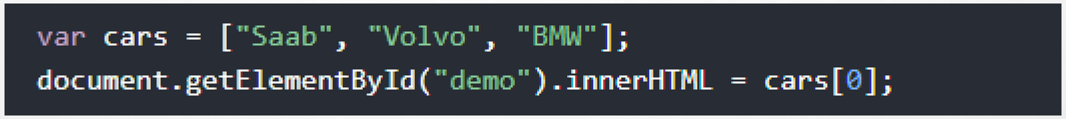
```
var cars = new Array("Saab", "Volvo", "BMW");
```

JAVASCRIPT ARRAYS

Accessing the element of an Array :

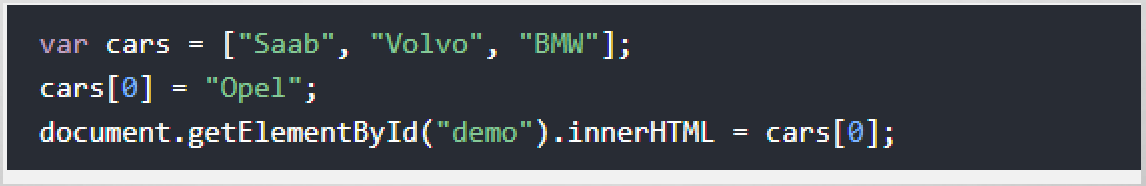
You access an array element by referring to the index number.

This statement accesses the value of the first element in cars:.



```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

Changing the array Element:



```
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars[0];
```

JAVASCRIPT METHODS

Method	**Description**
concat()	Joins two or more arrays, and returns a copy of the joined arrays
copyWithin()	Copies array elements within the array, to and from specified positions
entries()	Returns a key/value pair Array Iteration Object
every()	Checks if every element in an array pass a test
fill()	Fill the elements in an array with a static value
filter()	Creates a new array with every element in an array that pass a test
find()	Returns the value of the first element in an array that pass a test
findIndex()	Returns the index of the first element in an array that pass a test
forEach()	Calls a function for each array element
from()	Creates an array from an object
includes()	Check if an array contains the specified element
indexOf()	Search the array for an element and returns its position

JAVASCRIPT METHODS

Method	Description
isArray()	Checks whether an object is an array
join()	Joins all elements of an array into a string
keys()	Returns a Array Iteration Object, containing the keys of the original array
lastIndexOf()	Search the array for an element, starting at the end, and returns its position
map()	Creates a new array with the result of calling a function for each array element
pop()	Removes the last element of an array, and returns that element
push()	Adds new elements to the end of an array, and returns the new length
reduce()	Reduce the values of an array to a single value (going left-to-right)
reduceRight()	Reduce the values of an array to a single value (going right-to-left)
reverse()	Reverses the order of the elements in an array
shift()	Removes the first element of an array, and returns that element
slice()	Selects a part of an array, and returns the new array
some()	Checks if any of the elements in an array pass a test
sort()	Sorts the elements of an array
splice()	Adds/Removes elements from an array
toString()	Converts an array to a string, and returns the result
unshift()	Adds new elements to the beginning of an array, and returns the new length
valueOf()	Returns the primitive value of an array

JAVASCRIPT DATE OBJECTS

JavaScript Date Output :

By default, JavaScript will use the browser's time zone and display a date as a full text string:

Wed Nov 11 2020 15:26:19 GMT+0530 (India Standard Time)

Creating Date Objects

Date objects are created with the new Date() constructor.

There are 4 ways to create a new date object:

```
new Date()
new Date(year, month, day, hours, minutes, seconds, milliseconds)
new Date(milliseconds)
new Date(date string)
```

JAVASCRIPT DATE OBJECTS

```
var d = new Date();
```

```
var d = new Date(2018, 11, 24, 10, 33, 30, 0);
```

```
var d = new Date("October 13, 2014 11:13:00");
```

```
var d = new Date(100000000000);
```

JAVASCRIPT DATE GET METHODS

Method	**Description**
getFullYear()	Get the **year** as a four digit number (yyyy)
getMonth()	Get the **month** as a number (0-11)
getDate()	Get the **day** as a number (1-31)
getHours()	Get the **hour** (0-23)
getMinutes()	Get the **minute** (0-59)
getSeconds()	Get the **second** (0-59)
getMilliseconds()	Get the **millisecond** (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay()	Get the weekday as a number (0-6)
Date.now()	Get the time. ECMAScript 5.

JAVASCRIPT DATE SET METHODS

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

JAVASCRIPT CONDITIONAL STATEMENT

Conditional Statements

Conditional statements are used to perform different actions based on different conditions.

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

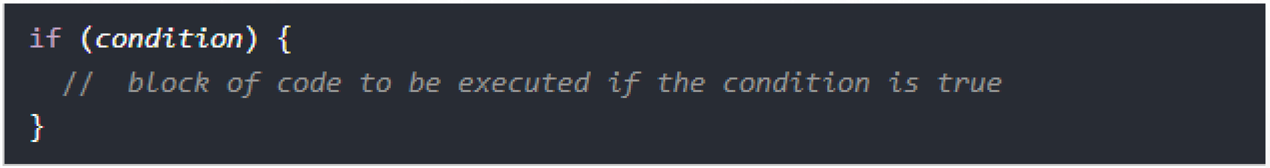
You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

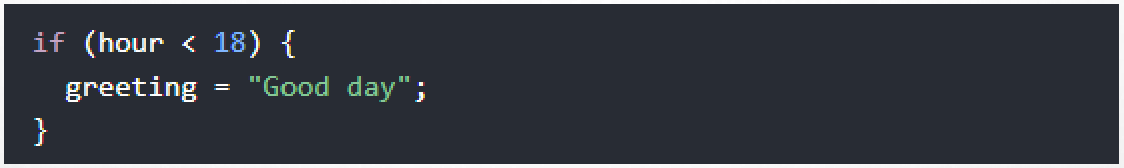
- Use **if** to specify a block of code to be executed, if a specified condition is true
- Use **else** to specify a block of code to be executed, if the same condition is false
- Use **else if** to specify a new condition to test, if the first condition is false
- Use **switch** to specify many alternative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

JAVASCRIPT CONDITIONAL STATEMENT

<u>If Statement</u>

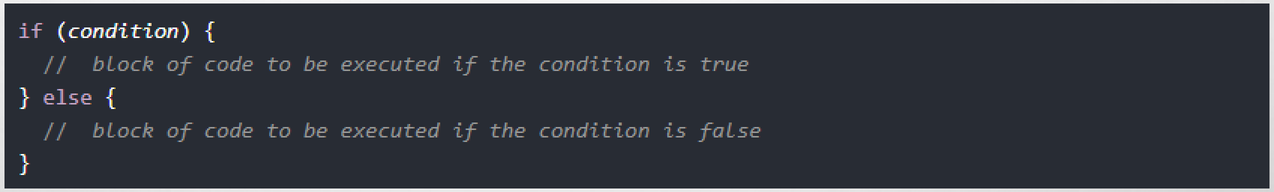


```
if (condition) {
  //  block of code to be executed if the condition is true
}
```

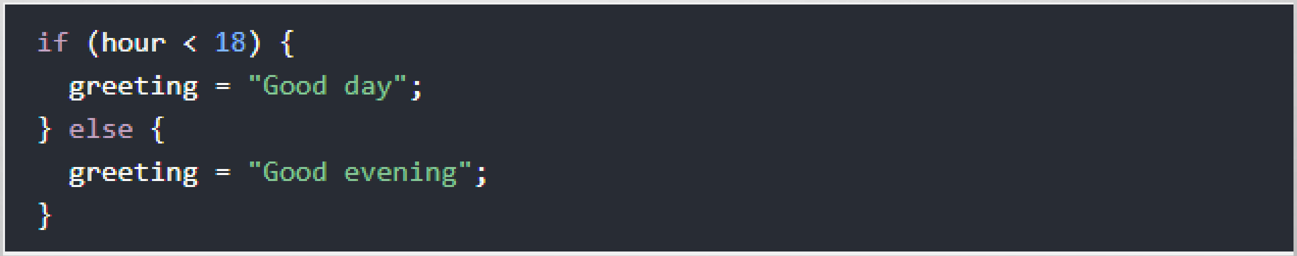


```
if (hour < 18) {
  greeting = "Good day";
}
```

<u>If else Statement</u>: Use the else statement to specify a block of code to be executed if the condition is false.



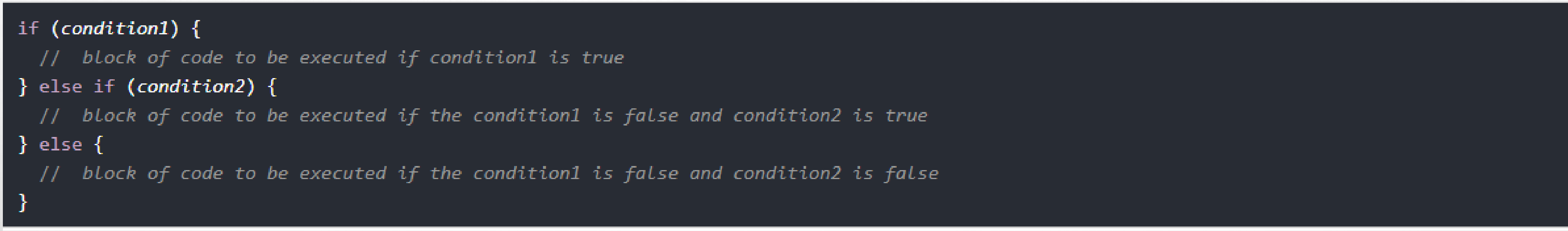
```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```



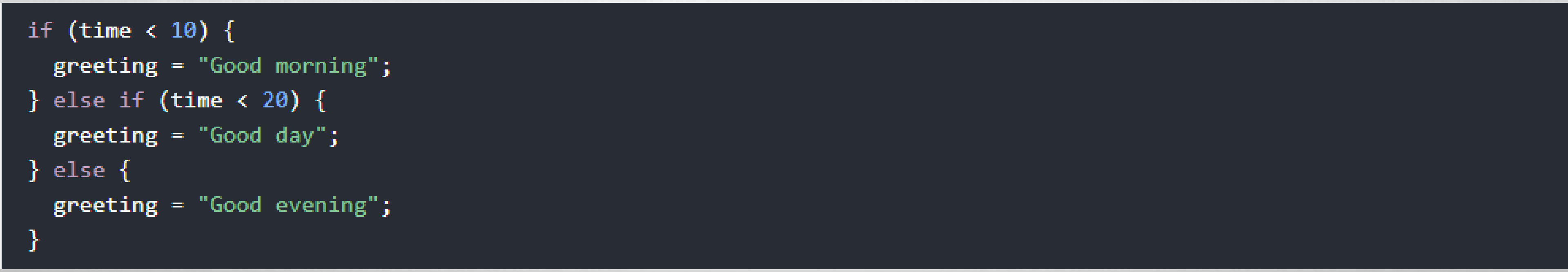
```
if (hour < 18) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

JAVASCRIPT CONDITIONAL STATEMENT

<u>The else If Statement</u> : Use the else if statement to specify a new condition if the first condition is false.



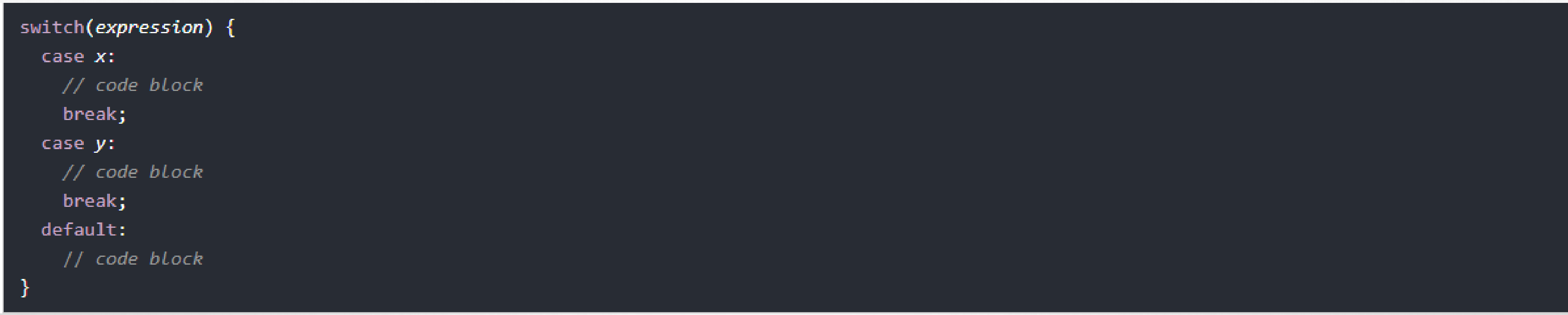
```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```



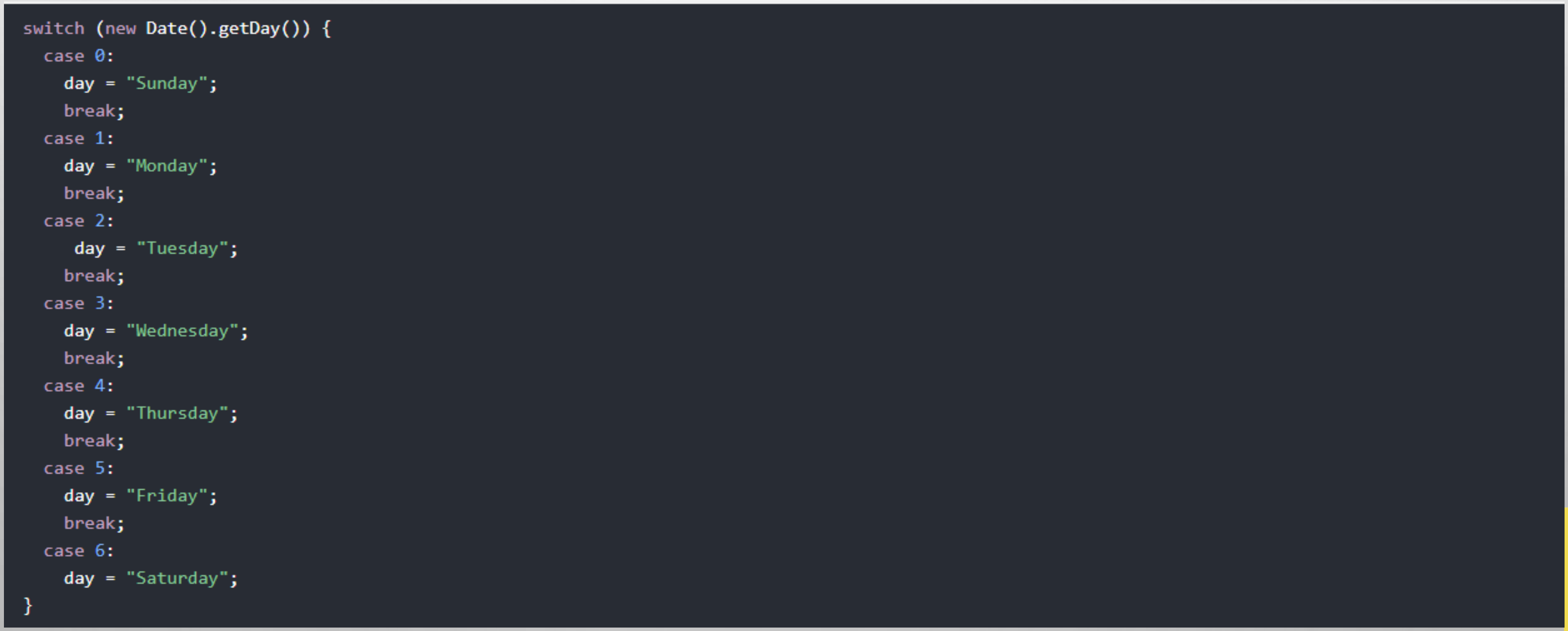
```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

JAVASCRIPT CONDITIONAL STATEMENT (SWITCH STATMENT)

<u>**Switch Statement**</u>: The switch statement is used to perform different actions based on different conditions.



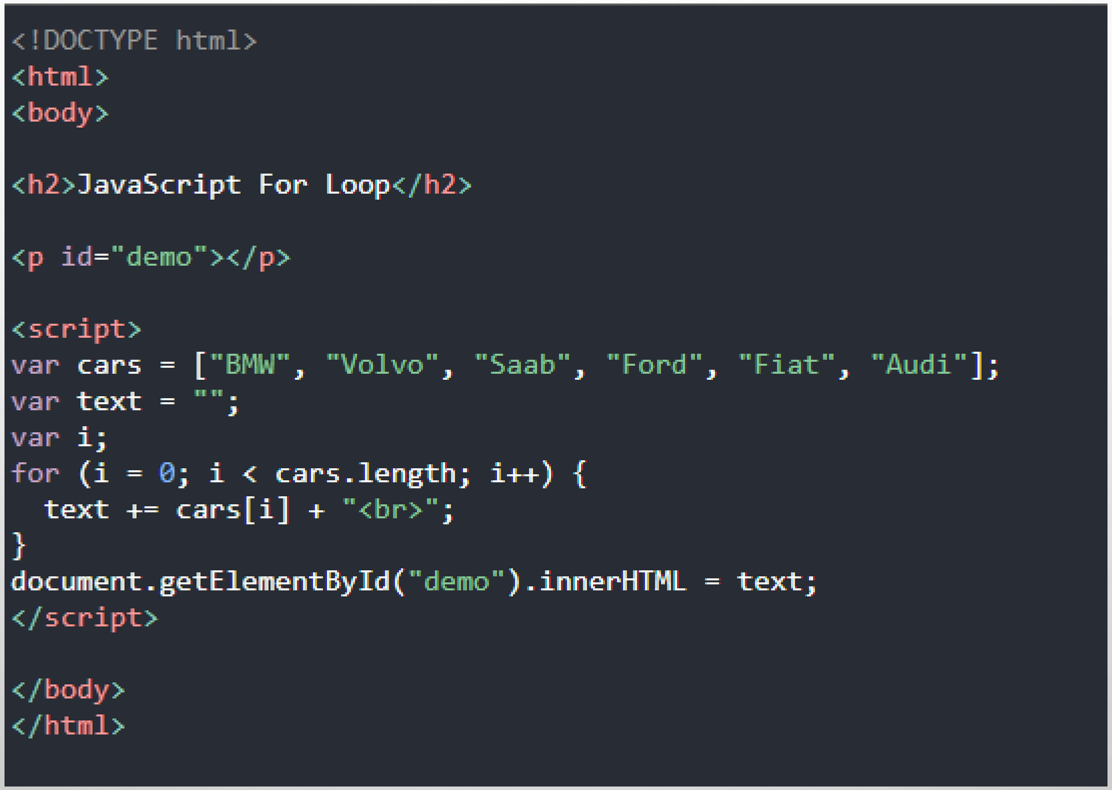
```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```



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

JAVASCRIPT FOR LOOP

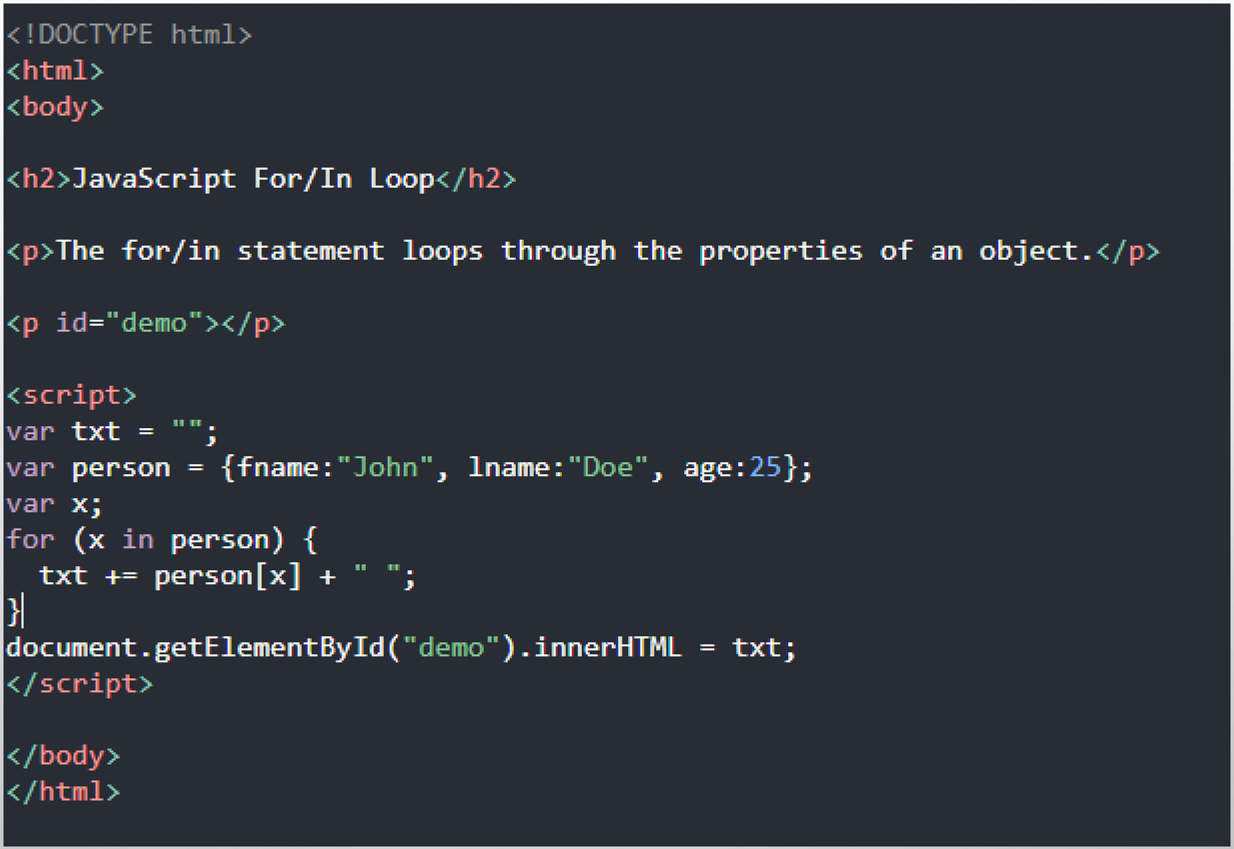
For Loop: Loops can execute a block of code a number of times.



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

<h2>JavaScript For Loop</h2>

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

<script>
var cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];
var text = "";
var i;
for (i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JAVASCRIPT FOR LOOP

<u>For/in Loop</u>: The JavaScript for/in statement loops through the properties of an object:



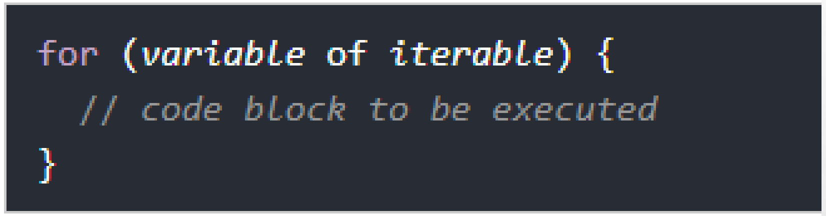
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For/In Loop</h2>

<p>The for/in statement loops through the properties of an object.</p>

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

<script>
var txt = "";
var person = {fname:"John", lname:"Doe", age:25};
var x;
for (x in person) {
  txt += person[x] + " ";
}
document.getElementById("demo").innerHTML = txt;
</script>

</body>
</html>
```

JAVASCRIPT FOR LOOP

For/of Loop: The JavaScript for/of statement loops through the values of an iterable objects.

for/of lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more.

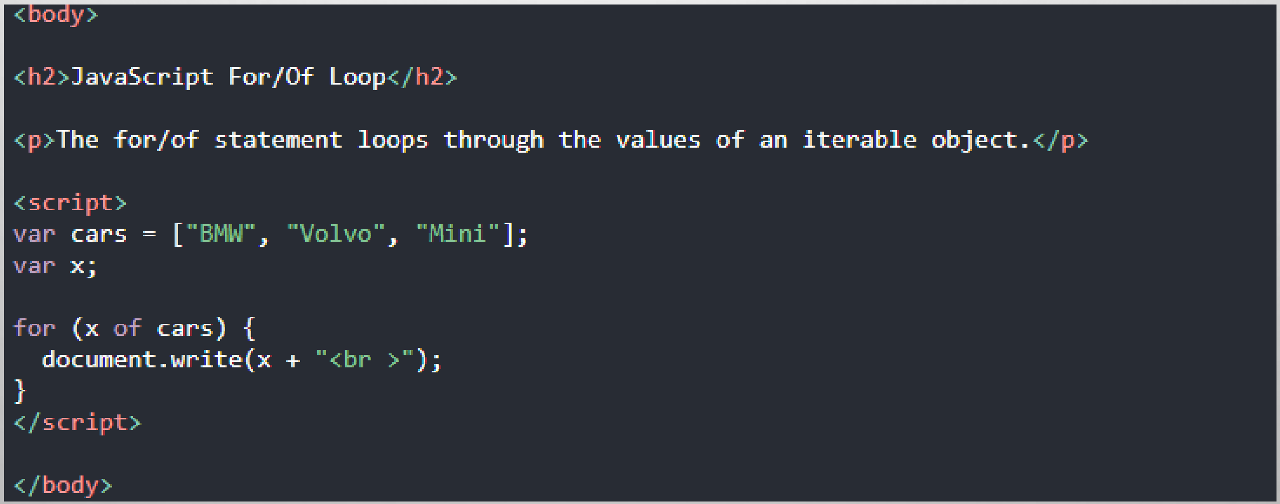
The for/of loop has the following syntax:



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

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

iterable - An object that has iterable properties.



```
<body>

<h2>JavaScript For/Of Loop</h2>

<p>The for/of statement loops through the values of an iterable object.</p>

<script>
var cars = ["BMW", "Volvo", "Mini"];
var x;

for (x of cars) {
  document.write(x + "<br >");
}
</script>

</body>
```

JAVASCRIPT WHILE LOOP

The While Loop

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

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

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

<h2>JavaScript While Loop</h2>

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

<script>
var text = "";
var i = 0;
while (i < 10) {
  text += "<br>The number is " + i;
  i++;
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JS

JAVASCRIPT DO/WHILE LOOP

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.

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

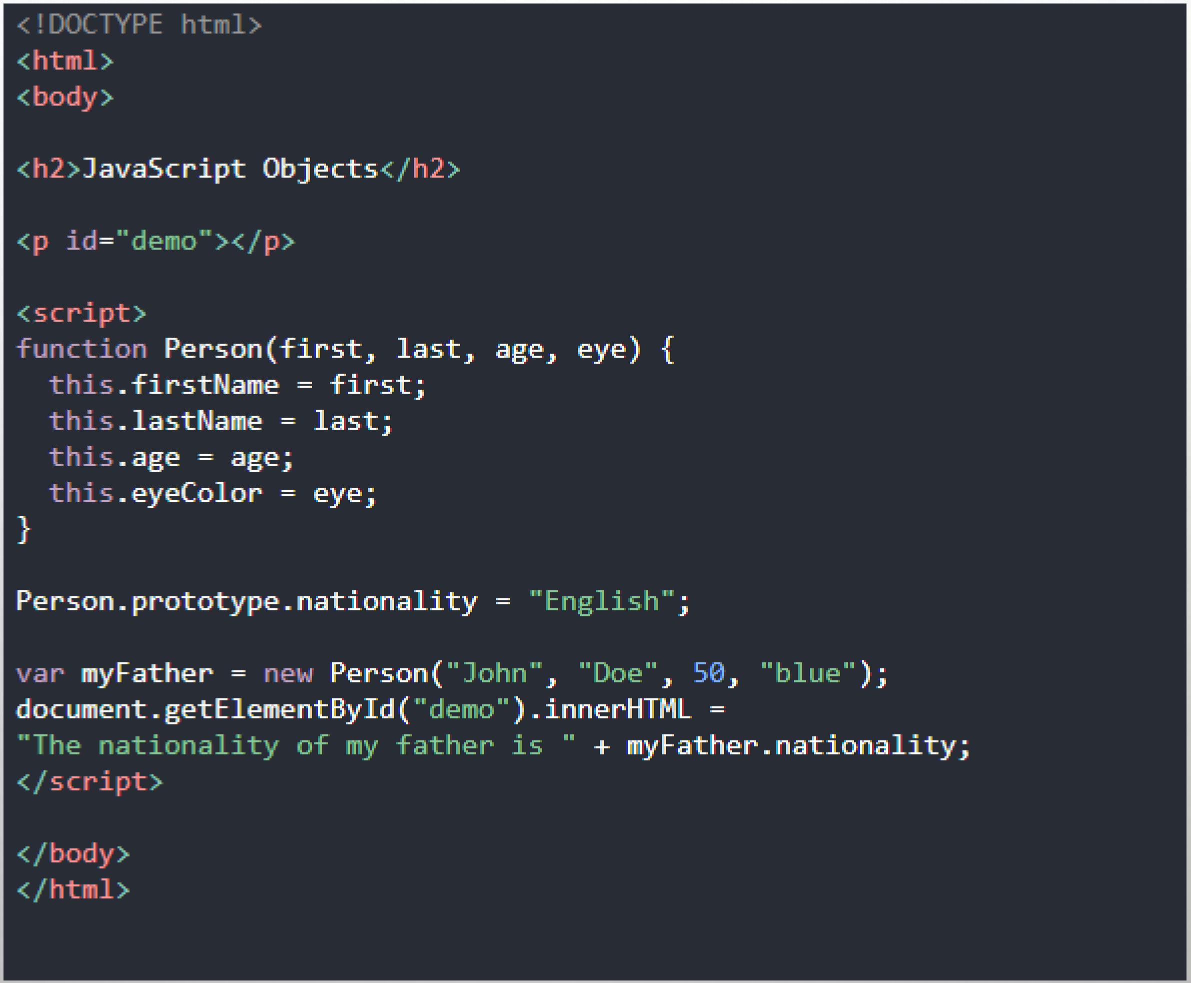
The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

```
<h2>JavaScript Do/While Loop</h2>

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

<script>
var text = ""
var i = 0;

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

document.getElementById("demo").innerHTML = text;
</script>
```

JAVASCRIPT OBJECT PROTOTYPE

JavaScript Object Prototype:

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



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

<h2>JavaScript Objects</h2>

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

<script>
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}

Person.prototype.nationality = "English";

var myFather = new Person("John", "Doe", 50, "blue");
document.getElementById("demo").innerHTML =
"The nationality of my father is " + myFather.nationality;
</script>

</body>
</html>
```

SELF-EXECUTING FUNCTION

Self-Executing function:

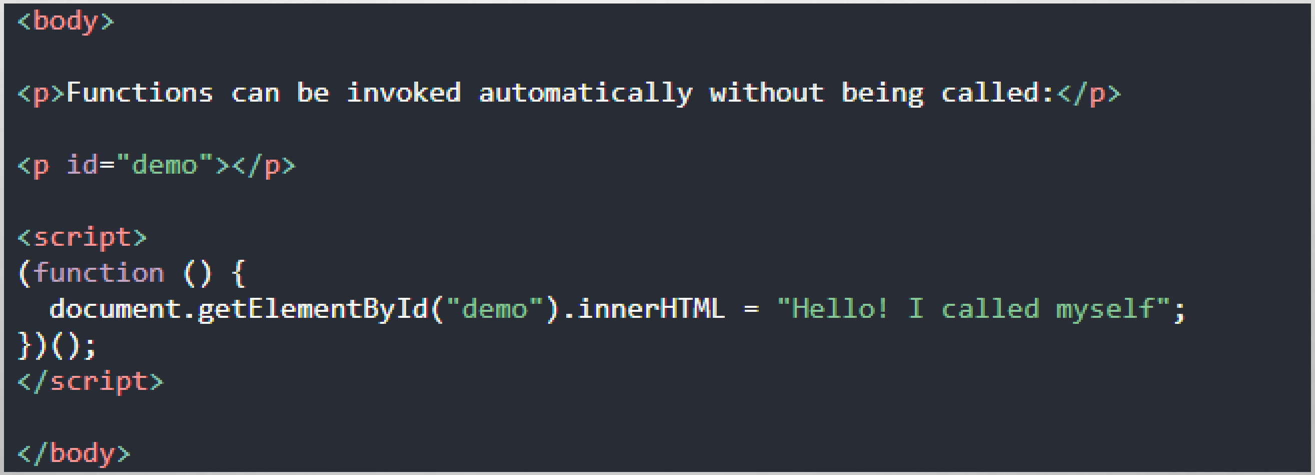
Function expressions can be made "self-invoking".

A self-invoking expression is invoked (started) automatically, without being called.

Function expressions will execute automatically if the expression is followed by ().

You cannot self-invoke a function declaration.

You have to add parentheses around the function to indicate that it is a function expression:



```
<body>

<p>Functions can be invoked automatically without being called:</p>

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

<script>
(function () {
  document.getElementById("demo").innerHTML = "Hello! I called myself";
})();
</script>

</body>
```