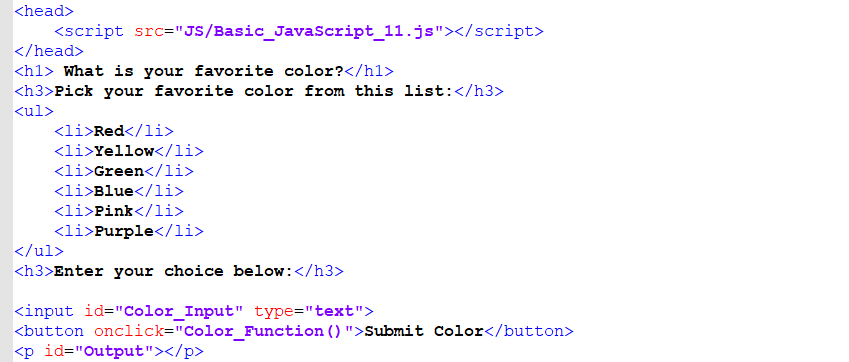
**SWITCH STATEMENT**

The switch statement performs different actions based on the conditions presented to it. It is a type of conditional statement.

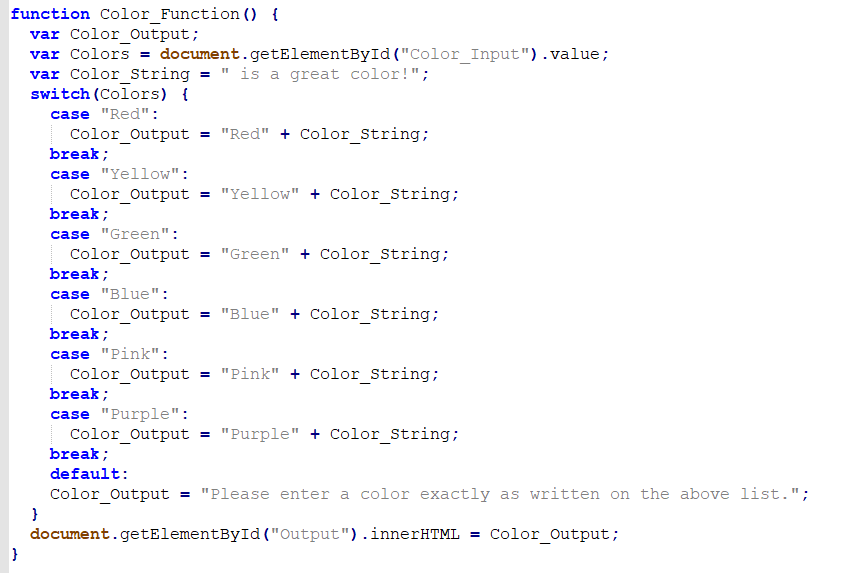
Inside switch statements are cases. These cases are the various conditions that are evaluated.

If there is no case match, the default keyword is used to specify what code to run. You can only utilize one default keyword per switch. It is considered a best practice to include it in your switch statements but it is not required.

We can use the switch statement to write a basic program that evaluates various colors . To start, we’d include the following HTML code in our template:



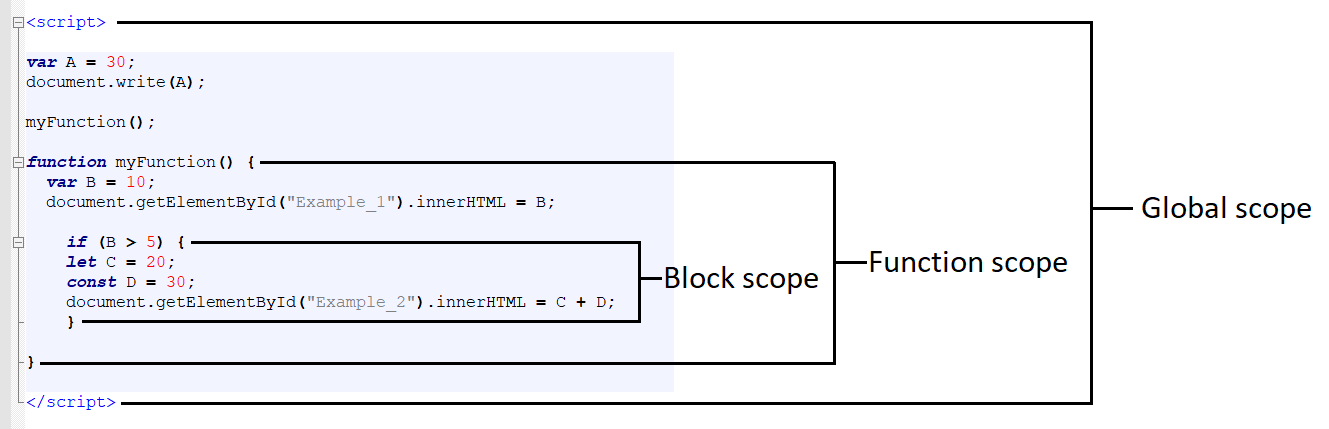
Then in our JavaScript code we’d write:



**SCOPES**

Whereas var can have function or global scope, let and const can have function, global or block scope.

Here is a diagram:



Global scope: The variable “A” would be able to be accessed from anywhere in the program.

Function scope: The variable “B” would only be able to be accessed from within the function – not outside.

Block scope: The variables “C” and “D” could only be accessed within the if statement.

For example: if you attempted to perform an operation on the variables C and D outside of the if statement’s curly brackets, it wouldn’t execute.

**STEP: 193**

Functions are useful so you don’t have to keep retyping code; you can instead simply invoke the function whenever needed.

An example could look like this:

https://techacademystorage.blob.core.windows.net/javascript/functions2.png

Other JavaScript code elements could call this “add” function by specifying its name and passing it two numbers. That could look like this:

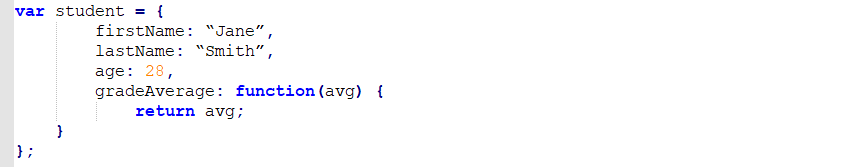
https://techacademystorage.blob.core.windows.net/javascript/functions3.png

Here, the code would create the variable called “sum”, call the function “add” and pass it the two numbers 5 and 7, and take the result (12) and assign that value to the variable “sum”.

**METHODS**

In JavaScript, a method is a set of code associated with an object that is designed to change the state of that object when it executes. In other words, the method is performed on the object.

You create these methods when you create the object. Let’s look at how we might do that with our previous example of a “student” object:



Here, we aren’t setting the property “gradeAverage” to a fixed number of 3.5. Instead, we are setting that property to the value returned by a set of code. Specifically, that code will take in a number (the variable “avg”) and set the value of the “gradeAverage” property to the value of that variable.

Executing that code could look like this:

https://techacademystorage.blob.core.windows.net/javascript/functions5.png

Here, we are telling the computer to make use of the object called “student”. Specifically, the computer is to run the function “gradeAverage” that is defined in the “student” object. Since that function needs an input (the variable “avg”), we give it the number 3.4.

So here’s the distinction: In this specific situation, that function “gradeAverage” is called a method. Yes, it’s confusing – until you recall that in JavaScript, a method is a set of code, associated with an object, that is performed on the object itself. So we have the confusing situation of a property of an object that we are defining using a function – but we’re calling the action performed by the function a method.

**LINKING JAVASCRIPT FILES**

Just like with CSS, our JavaScript code can be written in external files that can be linked to from our HTML file. To do so, we include the src attribute within a script tag, as follows:

https://techacademystorage.blob.core.windows.net/javascript/linking_javascript.png

So, to run an external file, we would write this code within the external file:

https://techacademystorage.blob.core.windows.net/javascript/linking_javascript2.PNG

Save the file as “JavaScript\_File.js.” Then, within the body or head text of our HTML document, we’d write:

https://techacademystorage.blob.core.windows.net/javascript/linking_javascript3.PNG

It is best practice to keep all JavaScript within an external file.

**ASSIGNMENT**

1. Create a new folder inside your Basic JavaScript Projects folder and name it “JS”.

2. Create a new file and write the alert() method.

3. Save this file as “Basic\_JavaScript\_1.js” within the JS folder.

4. Open your Basic\_HTML\_1.html and delete the script element.

5. Within the body or head write a script element containing a src attribute.

https://techacademystorage.blob.core.windows.net/javascript/linking_javascript4.PNG

6. Save the HTML file and then successfully execute it in the browser.

**WINDOW.ALERT() METHOD**

We can cause an alert window to pop up using the window.alert() method as well:

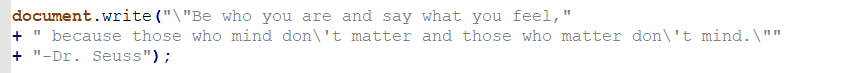
https://techacademystorage.blob.core.windows.net/javascript/methods1.png

**CONCATENATING A STRING**

“Concatenate” means to connect things together, like links in a chain. It means to take one piece of data and stick it on the end of another piece of data.

For example: concatenating the string “device” and the string “s” makes the text “devices.”

To concatenate a string in JavaScript, you use the + operator as follows:



This text would display as, "Be who you are and say what you feel, because those who mind don't matter and those who matter don't mind."-Dr. Seuss

**CHALLENGE**

Assign a concatenated string value to a variable and display it in the browser.

HINT:

https://techacademystorage.blob.core.windows.net/javascript/strings3.PNG

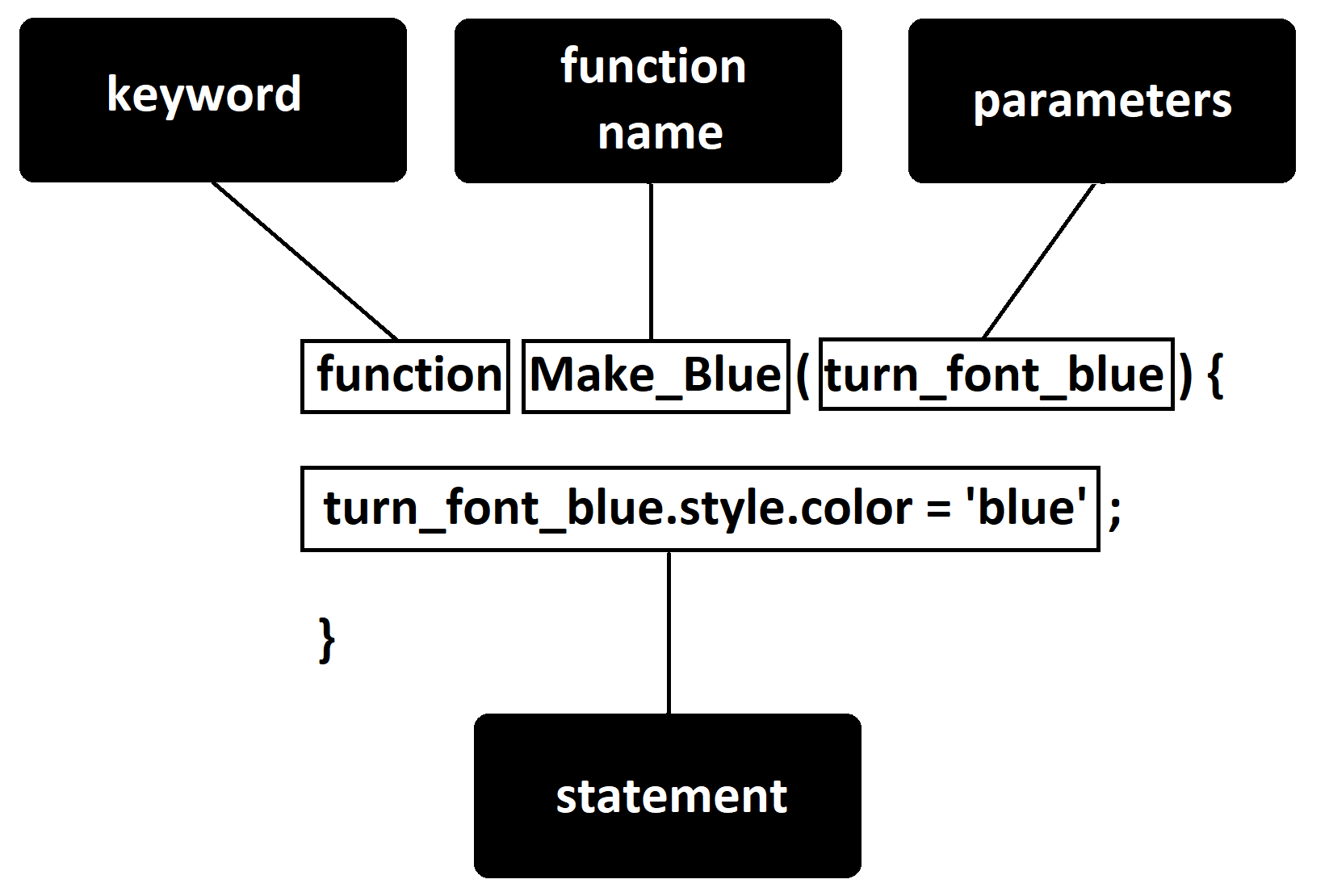
**FUNCTIONS**

As a recap, a JavaScript function is a repeatable block of code that executes certain actions and performs tasks. You execute a function by calling it. This is also called “invoking” the function.

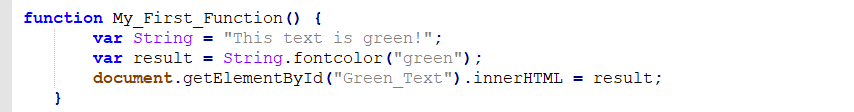
Functions are valuable because of code reusability – you can invoke functions over and over. Since functions in JavaScript contain properties and methods, they are basically objects.

“document.getElementById” is a method that returns an element. The element has an ID attribute with a specific value assigned to it. It is used mainly to control or get information from an element within your code. If it can’t find the element with the specified value, it will return “null.”

In JavaScript, a keyword identifies actions to be performed. There is a function keyword. Parameters are the values passed to or received by the function. One writes a JavaScript function with the keyword, then a name, then parentheses containing parameters (note: you can leave the parentheses empty).

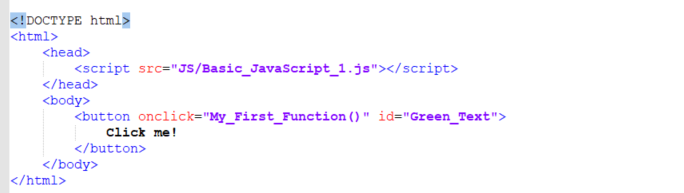


Here is an actual function (this is JavaScript code – it would be written within the external .js file):



Everything inside our curly brackets { } is our function.

The HTML code within our .html file would be written as follows:



In this HTML code, we created a button element. We then utilized the ID attribute and assigned the button element the value “Green\_Text.”

The button element was also given an event handler which is used to call the JavaScript function “My\_First\_Function()”. The onclick event handler is triggered when the user clicks on that element.

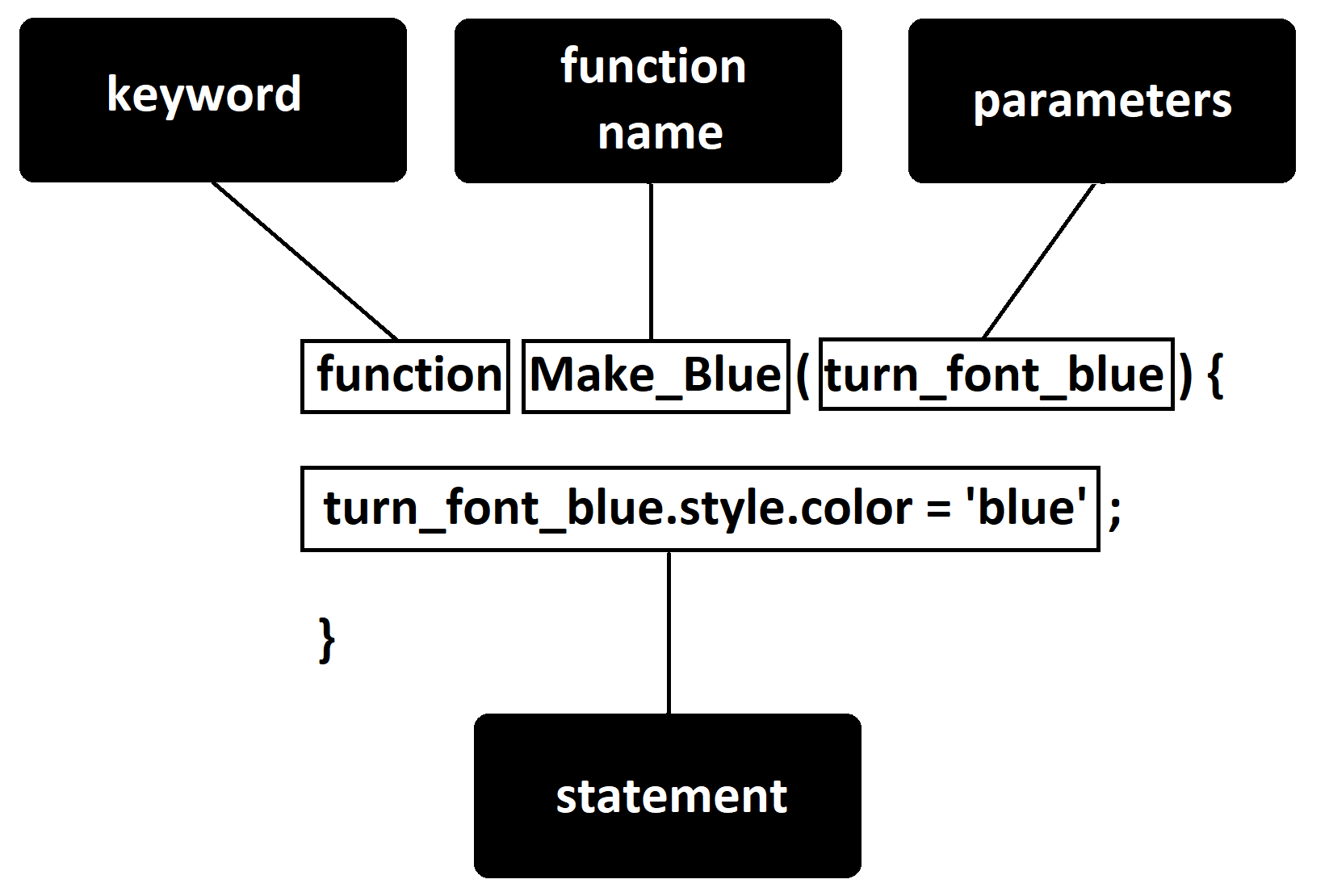
**FUNCTIONS**

As a recap, a JavaScript function is a repeatable block of code that executes certain actions and performs tasks. You execute a function by calling it. This is also called “invoking” the function.

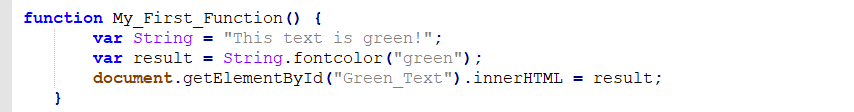
Functions are valuable because of code reusability – you can invoke functions over and over. Since functions in JavaScript contain properties and methods, they are basically objects.

“document.getElementById” is a method that returns an element. The element has an ID attribute with a specific value assigned to it. It is used mainly to control or get information from an element within your code. If it can’t find the element with the specified value, it will return “null.”

In JavaScript, a keyword identifies actions to be performed. There is a function keyword. Parameters are the values passed to or received by the function. One writes a JavaScript function with the keyword, then a name, then parentheses containing parameters (note: you can leave the parentheses empty).

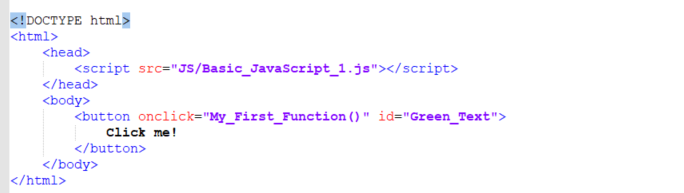


Here is an actual function (this is JavaScript code – it would be written within the external .js file):



Everything inside our curly brackets { } is our function.

The HTML code within our .html file would be written as follows:



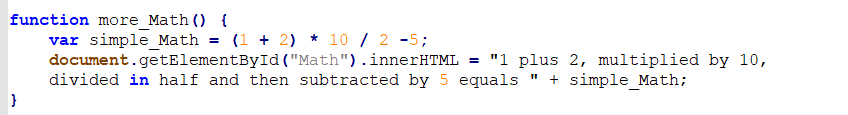
In this HTML code, we created a button element. We then utilized the ID attribute and assigned the button element the value “Green\_Text.”

The button element was also given an event handler which is used to call the JavaScript function “My\_First\_Function()”. The onclick event handler is triggered when the user clicks on that element.

Suppose you wanted to find all paragraph elements in an HTML document. You might use code like this:



The variable called “foo” would now contain a collection of all the paragraph tags in the document.

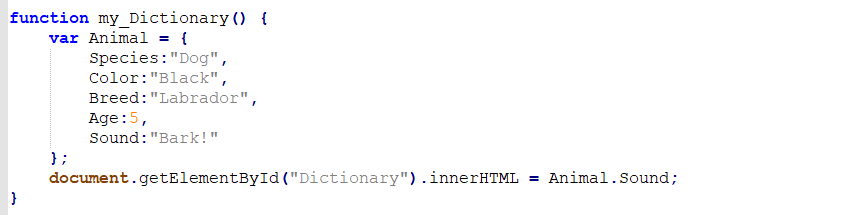


To create a dictionary in JavaScript you would do the following:



We would start by adding this to our HTML code:

Then in our JavaScript file, we would write:



The output of our code would be: Bark!

Because this was associated with the key “Sound.”

NOTE: Because 5 is the value type “number”, we don’t have to place it within quotation marks.

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

In JavaScript, almost "everything" is an object.

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

All JavaScript values, except primitives, are objects.

JavaScript Primitives

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

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

JavaScript defines 5 types of primitive data types:

* string
* number
* boolean
* null
* undefined

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

if x = 3.14, you can change the value of x. But you cannot change the value of 3.14.

|  |  |  |
| --- | --- | --- |
| **Value** | **Type** | **Comment** |
| "Hello" | string | "Hello" is always "Hello" |
| 3.14 | number | 3.14 is always 3.14 |
| true | boolean | true is always true |
| false | boolean | false is always false |
| null | null (object) | null is always null |
| undefined | undefined | undefined is always undefined |

## Objects are Variables

JavaScript variables can contain single values:

### **Example**

var person = "John Doe";

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

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

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

### **Example**

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

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

A JavaScript object is a collection of **named values**

## Object Properties

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

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

Objects written as name value pairs are similar to:

* Associative arrays in PHP
* Dictionaries in Python
* Hash tables in C
* Hash maps in Java
* Hashes in Ruby and Perl

## Object Methods

Methods are **actions** that can be performed on objects.

Object properties can be both primitive values, other objects, and functions.

An **object method** is an object property containing a **function definition**.

|  |  |
| --- | --- |
| **Property** | **Value** |
| firstName | John |
| lastName | Doe |
| age | 50 |
| eyeColor | blue |
| fullName | function() {return this.firstName + " " + this.lastName;} |

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

You will learn more about methods in the next chapters.

## Creating a JavaScript Object

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

There are different ways to create new objects:

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

In ECMAScript 5, an object can also be created with the function Object.create().

## Using an Object Literal

This is the easiest way to create a JavaScript Object.

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

An object literal is a list of name:value pairs (like age:50) inside curly braces {}.

The following example creates a new JavaScript object with four properties:

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_create_1)

Spaces and line breaks are not important. An object definition can span multiple lines:

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_create_2)

## Using the JavaScript Keyword new

The following example also creates a new JavaScript object with four properties:

### **Example**

var person = new Object();  
person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";

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

The two examples above do exactly the same. There is no need to use new Object().  
For simplicity, readability and execution speed, use the first one (the object literal method).

## JavaScript Objects are Mutable

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

If person is an object, the following statement will not create a copy of person:

var x = person;  // This will not create a copy of person.

The object x is **not a copy** of person. It **is** person. Both x and person are the same object.

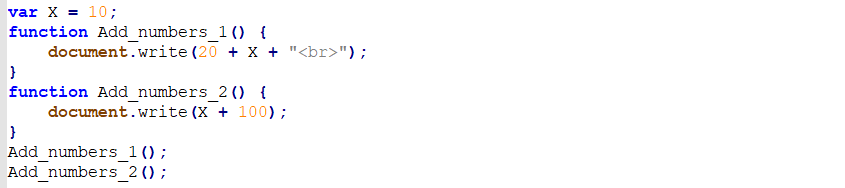
Any changes to x will also change person, because x and person are the same object.

### **Example**

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}  
  
var x = person;  
x.age = 10;           // This will change both x.age and person.age

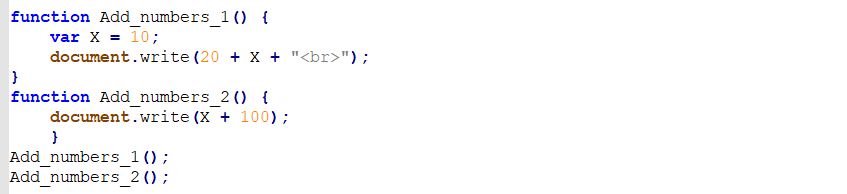
[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_mutable)

A global variable would be written as follows:



This code would return “30” and “110.” The variable X was assigned the value 10 outside of our function, but we still accessed it – therefore, the above is an example of a global variable.

The following would be an example of a local variable:



This time, the code would only return “30” because the variable was local – meaning it was written within the function Add\_numbers\_1 and couldn’t be accessed outside of it.

Let’s say you wrote the above code and didn’t understand why Add\_numbers\_2 didn’t display a result. We could use the console.log() method to help us debug our code as follows:

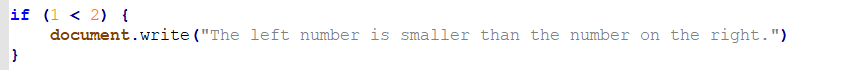


If you executed this code in the browser, no result will be shown. But if you open the console, you’ll see the error “X is not defined.”

**IF STATEMENTS**

“If statements” are a type of conditional statement that specifies that a section of code is to be executed if a condition is true.

For example:

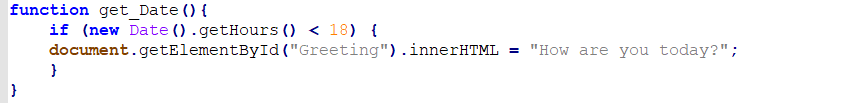


And since 1 is smaller than 2, the statement (“The left number is smaller than the number on the right.”) will display.

Here is another example – first we would write this within our HTML file:



Then in our JavaScript code, we’d write:



As a note: The Date().getHours() method returns the hour in the specified date according to the local time, and the hours are listed as integers between 0 and 23. For example: 18 is equal to 6:00 p.m., 12 is noon, etc.

Here is a list of JavaScript Get Date Methods:

[**https://www.w3schools.com/js/js\_date\_methods.asp**](https://www.w3schools.com/js/js_date_methods.asp)

In the above code we stated that if it is later than (greater than) 6:00 p.m., display “How are you this evening?” If you wrote this code earlier than 6:00 p.m. in the day, “How are you today?” will display

Global vs. Local:

<!DOCTYPE html>

<html>

<head>

<script src="JS/Basic\_JavaScript\_7.js"></script>

</head>

<body>

<p>Create a local varible, assign a value to it, and display it:</p>

<button type="button" onclick="myFunction()">Local Var!</button>

<p id="local\_var"></p>

<script>

function myFunction() {

var a = 5;

document.getElementById("local\_var").innerHTML = a \* a;

}

</script>

<p>Create a GLOBAL varible, assign a value to it, and display it:</p>

<button type="button" onclick="myFunction2()">Global Var!</button>

<p id="global\_var"></p>

<script>

var b = 5;

function myFunction2() {

document.getElementById("global\_var").innerHTML = b \* b;

}

</script>