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).

Example

// Function to compute the product of p1 and p2  
function myFunction(p1, p2) {  
  return p1 \* p2;  
}

JavaScript 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: **{}**

function *name*(*parameter1, parameter2, parameter3*) {  
  // *code to be executed*  
}

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

* When an event occurs (when a user clicks a button)
* When it is invoked (called) from JavaScript code
* Automatically (self invoked)

You will learn a lot more about function invocation later in this tutorial.

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":

Example

Calculate the product of two numbers, and return the result:

// Function is called, the return value will end up in x  
let x = myFunction(4, 3);  
  
function myFunction(a, b) {  
// Function returns the product of a and b  
  return a \* b;  
}

Why Functions?

With functions you can reuse code

You can write code that can be used many times.

You can use the same code with different arguments, to produce different results.

The () Operator

The () operator invokes (calls) the function:

Example

Convert Fahrenheit to Celsius:

function toCelsius(fahrenheit) {  
  return (5/9) \* (fahrenheit-32);  
}  
  
let value = toCelsius(77);

Accessing a function with incorrect parameters can return an incorrect answer:

Example

function toCelsius(fahrenheit) {  
  return (5/9) \* (fahrenheit-32);  
}  
  
let value = toCelsius();

Accessing a function without () returns the function and not the function result:

Example

function toCelsius(fahrenheit) {  
  return (5/9) \* (fahrenheit-32);  
}  
  
let value = toCelsius;

Note

As you see from the examples above, toCelsius refers to the function object, and toCelsius() refers to the function result.

Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

Example

Instead of using a variable to store the return value of a function:

let x = toCelsius(77);  
let text = "The temperature is " + x + " Celsius";

You can use the function directly, as a variable value:

let text = "The temperature is " + toCelsius(77) + " Celsius";

You will learn a lot more about functions later in this tutorial.

Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.

Local variables can only be accessed from within the function.

Example

// code here can NOT use carName  
  
function myFunction() {  
  let carName = "Volvo";  
  // code here CAN use carName  
}  
  
// code here can NOT use carName

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

# JavaScript Object Methods

**Object methods** are actions that can be performed on objects.

A method is a **function definition** stored as a **property value**.

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

### Example

const **person** = {  
  firstName: "John",  
  lastName: "Doe",  
  id: 5566,  
  fullName: function() {  
    return **this**.firstName + " " + **this**.lastName;  
  }  
};

In the example above, this refers to the **person object**:

**this.firstName** means the **firstName** property of **person**.

**this.lastName** means the **lastName** property of **person**.

## Accessing Object Methods

You access an object method with the following syntax:

*objectName.methodName()*

If you invoke the **fullName property** with (), it will execute as a **function**:

### Example

name = person.fullName();

If you access the **fullName property** without (), it will return the **function definition**:

### Example

name = person.fullName;

## Adding a Method to an Object

Adding a new method to an object is easy:

### Example

person.name = function () {  
  return this.firstName + " " + this.lastName;  
};

## Using JavaScript Methods

This example uses the JavaScript toUpperCase() method to convert a text to uppercase:

### Example

person.name = function () {  
  return (this.firstName + " " + this.lastName).toUpperCase();  
};

# JavaScript Object Constructors

## Object Constructor Functions

Sometimes we need to create many objects of the same **type**.

To create an **object type** we use an **object constructor function**.

It is considered good practice to name constructor functions with an upper-case first letter.

### Object Type Person

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

Now we can use new Person() to create many new Person objects:

### Example

const myFather = new Person("John", "Doe", 50, "blue");  
const myMother = new Person("Sally", "Rally", 48, "green");  
const mySister = new Person("Anna", "Rally", 18, "green");  
  
const mySelf = new Person("Johnny", "Rally", 22, "green");

## Property Default Values

A **value** given to a property will be a **default value** for all objects created by the constructor:

### Example

function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
  this.nationality = "English";  
}

## Adding a Property to an Object

Adding a property to a created object is easy:

### Example

myFather.nationality = "English";

## Adding a Property to a Constructor

You can **NOT** add a new property to an object constructor:

### Example

Person.nationality = "English";

To add a new property, you must add it to the constructor function prototype:

### Example

Person.prototype.nationality = "English";

## Constructor Function Methods

A constructor function can also have **methods**:

### Example

function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
  this.fullName = function() {  
    return this.firstName + " " + this.lastName;  
  };  
}

## Adding a Method to an Object

Adding a method to a created object is easy:

### Example

myMother.changeName = function (name) {  
  this.lastName = name;  
}

## Adding a Method to a Constructor

You cannot add a new method to an object constructor function.

This code will produce a TypeError:

### Example

Person.changeName = function (name) {  
  this.lastName = name;  
}  
  
myMother.changeName("Doe");

 TypeError: myMother.changeName is not a function

Adding a new method must be done to the constructor function prototype:

### Example

Person.prototype.changeName = function (name) {  
  this.lastName = name;  
}  
  
myMother.changeName("Doe");

## Built-in JavaScript Constructors

JavaScript has built-in constructors for all native objects:

new Object()   // A new Object object  
new Array()    // A new Array object  
new Map()      // A new Map object  
new Set()      // A new Set object  
new Date()     // A new Date object  
new RegExp()   // A new RegExp object  
new Function() // A new Function object