1--->

Use this article as a reference sheet for JavaScript methods and functions.

Function — a set of instructions that perform a task.

Method — a set of instructions that are associated with an object.

FUNCTIONS

Functions are like recipes. They can execute a set of instructions on data or variables and return the result. The beauty of functions is that they are recyclable. That is, the function can be used repeatedly without having to write the same code again.

// Define a function that prints a string

function welcomeMessage() {

console.log('Welcome to JavaScript');

}

// Call the function

welcomeMessage();

In the example above, the welcomeMessage function is used to display Welcome to JavaScript in the console. Let’s walk through this code step-by-step:

The function keyword indicates the start of a function.

The word that follows (welcomeMessage) is the function’s name.

The empty parentheses after welcomeMessage indicate that there are no parameters, or inputs, for the function.

The code between the opening ({) and closing (}) curly braces is a set of instructions. This code will only execute when the function is called.

To call a method, you need the function’s name, a pair of parentheses, and a semicolon. In this example, the function is called with welcomeMessage();.

Let’s also consider a function that receives inputs and returns outputs:

function concatName(firstName, middleName, lastName) {

return firstName + ' ' + middleName + ' ' + lastName;

}

Let’s walk through this example step-by-step:

The concatName function is created with three parameters (inputs): firstName, middleName, and lastName. Think of these as variables that have not been set yet.

Inside of the function, the return keyword will return the concatenation of firstName, middleName, and lastName, with spaces in between each. The return keyword terminates the function it is within, and returns a value — any code inside the function that comes after the return keyword will not be evaluated (nor executed, itself).

You can save the returned string to a variable or log it to the console when you call the function.

var concatGWC = concatName('George', 'Washington', 'Carver');

In the example above, the concatName function is called with the following arguments: 'George', 'Washington', and 'Carver'. These values are saved into the variables firstName, middleName, and lastName inside of the function. The function returns a concatenation of these three strings — the concatGWC variable stores the returned concatenated string.

In this reference sheet we have used the words parameters and arguments to reference similar, but distinctly different elements of a function and function call. What’s the difference between these words?

Parameters are fields that serve as variable names inside of a function. In the example above, firstName, middleName, and lastName are parameters.

Arguments are the values passed to the function when it is called. In the example above, 'George', 'Washington', and 'Carver' are arguments.

METHODS

A method, like a function, is a set of instructions that perform a task. The difference is that a method is associated with an object, while a function is not. Let’s explore some of JavaScript’s built-in methods.

var str = 'CodeCADEMY';

var str1 = str.toLowerCase();

var str2 = str.toUpperCase();

The variable str in the example above stores the string ‘CodeCADEMY’. The .toLowerCase() and .toUpperCase() methods in the example above are called on str. Let’s talk through each line:

On the second line, the .toLowerCase() method is called on the str variable, which returns the lowercase string 'codecademy'.

On the third line, the .toUpperCase() method is called on the str variable, which returns the uppercase string 'CODECADEMY'. Notice, periods are used to call a method on an object, as in str.toLowerCase();.

2--->

Var

The JavaScript variables statement is used to declare a variable and, optionally, we can initialize the value of that variable.

Example: var a =10;

Variable declarations are processed before the execution of the code.

The scope of a JavaScript variable declared with var is its current execution context.

The scope of a JavaScript variable declared outside the function is global.

Consider the following code snippet.

function nodeSimplified(){

var a =10;

console.log(a); // output 10

if(true){

var a=20;

console.log(a); // output 20

}

console.log(a); // output 20

}

In the above code, you can find, when the variable is updated inside the if loop, that the value of variable "a" updated 20 globally, hence outside the if loop the value persists. It is similar to the Global variable present in other languages. But, be sure to use this functionality with great care because there is the possibility of overriding an existing value.

let

The let statement declares a local variable in a block scope. It is similar to var, in that we can optionally initialize the variable.

Example: let a =10;

The let statement allows you to create a variable with the scope limited to the block on which it is used.

It is similar to the variable we declare in other languages like Java, .NET, etc.

Consider the following code snippet.

function nodeSimplified(){

let a =10;

console.log(a); // output 10

if(true){

let a=20;

console.log(a); // output 20

}

console.log(a); // output 10

}

It is almost the same behavior we see in most language.

function nodeSimplified(){

let a =10;

let a =20; //throws syntax error

console.log(a);

}

Error Message: Uncaught SyntaxError: Identifier 'a' has already been declared.

However, with var, it works fine.

function nodeSimplified(){

var a =10;

var a =20;

console.log(a); //output 20

}

The scope will be well maintained with a let statement and when using an inner function the let statement makes your code clean and clear.

I hope the above examples will help you better understand the var and let commands and if you have any queries please write me in the comment section.

const

const statement values can be assigned once and they cannot be reassigned. The scope of const statement works similar to let statements.

Example: const a =10;

function nodeSimplified(){

const MY\_VARIABLE =10;

console.log(MY\_VARIABLE); //output 10

}

As per usual, naming standards dictated that we declare the const variable in capital letters. const a =10 will work the same way as the code given above. Naming standards should be followed to maintain the code for the long run.

3--->

The Concept of Data Types

In programming, data types is an important concept.

To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

var x = 16 + "Volvo";

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

var x = "16" + "Volvo";

When adding a number and a string, JavaScript will treat the number as a string.

Example

var x = 16 + "Volvo";