college CDI

# Scripting language I (LS1)

**Course 7**

# Functions

A function is a sort of subprogram executing a group of instructions when called within a main program or in a page. One major benefit of using functions is that they make it possible to execute sometimes long sequences of instruction simply by calling them.

JavaScript offers many predefined functions (native objects). In fact, these are called methods (we used the term function to simplify). We already used a few such as *alert()*, *prompt()*, *confirm()* and *parseInt()*.

Basically, try to imagine the main program formed of several blocks executing sequences of instructions. These blocks could be functions called within the main program.

## Defining a function

Before using a function, it must of course be defined. It must be given a name and instructions to execute. It will then be possible to call a function using its name for all of its instructions to be executed.

### Syntax of a function

```
function myFonction() {
        instructions;
}
```

**Explanation :**

1. The term *function* indicates the creation of a new function.
2. The chosen name of the function is followed by parenthesis.
3. Opening brace bracket indicates the start of the instructions.
4. Instructions between brace brackets.
5. Ending brace bracket indicates the end of the instructions and of the function.

**Note :** Functions can be named using letters, numbers and the underscore and dollar symbols. It must start with a letter and never include blank spaces.

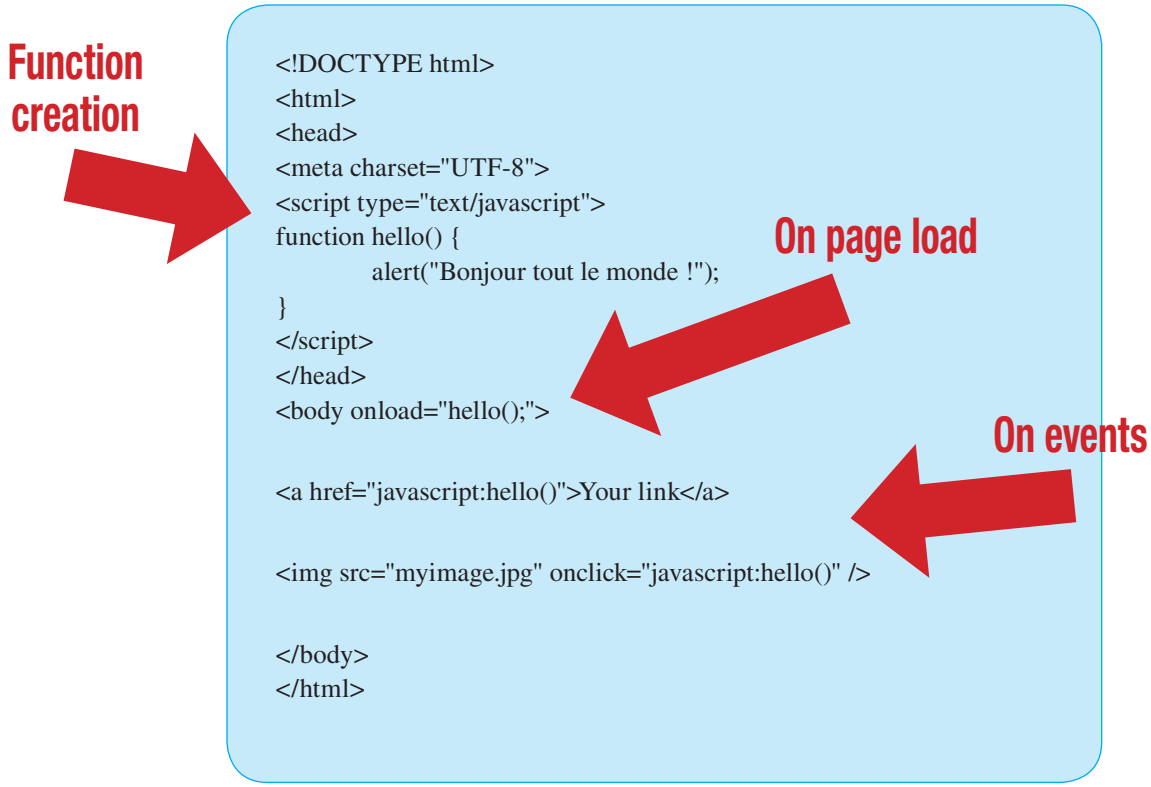
Also, remember JavaScript is case sensitive.

## Calling a function

Once a function has been created, it is necessary to call it from the main program in order to execute it. This can be made different ways :

- From the main program, another function, the function itself...
- On loading or unloading of a the page.
- Using events different events.
- Upon verification of a condition, etc..

**Examples of how a function can be called :**




```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
function hello() {
        alert("Bonjour tout le monde !");
}
</script>
</head>
<body onload="hello();">

<a href="javascript:hello()">Your link</a>

<img src="myimage.jpg" onclick="javascript:hello()" />

</body>
</html>
```

### Exercise 5

Create a function that will be called on click of a button.

The surface area of a rectangle will be supplied after user will have specified the width and the height of a rectangle.

**Here's the formula to calculate the surface area of a square/rectangle :**

**width x lenght = surface area**

## Availability of variables

It is important to remember that variables created within a function are not available in the main program (they are ***local variables***). But, variables created outside of the functions (in the main program) can be used everywhere, even within functions (they are ***global variables***).

However, it would be a bad practice to declare all variables as global in the main program since they would be accessible everywhere. Therefor, declare global variables when they are going to be needed in different places, and use local variables when these won't.

## Returned values and arguments

Returned values and arguments make it possible for functions to communicate with the main program. An argument is an object, an instruction and they are used to pass informations to functions and values are results returned by functions.

### Creating and using arguments

We have been using arguments since the very beginning of this JavaScript course. Whenever we were adding a string to an alert() method, it was an argument we were creating.

***alert (without an argument) :***

```
alert();
```

***alert (with an argument) :***

```
alert("My message to be displayed");
```

```
function myFunction(arg) {
        alert("Your argument is : " + arg);

/*  When the argument is passed to the function
    it is stored within the variable « arg » */

}
myFunction("Wow! I passed an argument!");
```

**Explanation:**

The function create a message window displaying a string followed by a variable (arg). Adding a string (also called chain) to *myFonction*, it is then associated to the variable.

### Defining the value to be passed to a function using prompt

```
function myFunction(arg) {
        alert("Your argument is : " + arg); ;
}
myFunction(prompt('What argument do you want to pass to this function?'));
```

**Explanation** :

The function *myFunction()* is first declared, the instructions (arguments) are stored in memory, but will be executed only upon calling the function.

The last instruction of this small program calls the function *myFunction()*, and passes an argument to it (prompt). Before being executed, the function must gather all necessary arguments ; here, one only : the user's answer.

The method *prompt()* is executed and then passes the value (string created by the user) to the function which can now be executed.

**Arguments availability**

Arguments passed to functions are stored in local variables; it can then be used like any other variables. Although, these variables are destroyed after the function has been executed.

**Note :** A colon is used at the end of the function's calling as it is considered an instruction in the main program.

### Passing multiple arguments

```
function myTest(first, second) {
        alert("First argument : " + first);
        alert("Second argument : " + second);
}
myTest(
        prompt("Write the first argument :"),
        prompt("Write the second argument :")
);
```

## Return values

Return values are those passed from functions to the main program or to other functions. These values are usually stored into variables.

Functions can basically return only one value (we will later see how to partially trick it so it can passes more).

For a function to return a value, the instruction *return* must be used followed by the value to be returned:

**Explanation:**

```
function sayHello() {
        return "Hello !";
}
alert(sayHello());
```

The instruction *return* followed by the string value « Hello ! » will return the value to the function which will display the message in an alert window.

**Note:** The intruction *return* ends (stops) the function. Instructions coded after return will be ignored.

## Anonymous functions

These functions, very important in JavaScript, serve many purposes. Although we won't start using them right away, it's a good moment to starting start them.

Stored into variables, these functions are called anonymous because they have no names. It's the only difference. The declaration of anonymous functions is done the exact same way as regular functions, although they aren't given names.

### Assigning function to a variable

```
function (argument) {
        instructions;
}
```

**Example :**

```
<script>
let sayHello = function() {
        alert('Hello !');
};

sayHello();
</script>
```

**Explanation:**

In the example, the anonymous function is stored into the variable *sayHello*. It is then a little like the variable has become a function, and it is used as one afterward.

### Isolating code

Isolating parts of source code can make it possible for one part of the code to influence the rest of the program. In order to do so, all that's needed is to place the part of the code to be isolated between parenthesis.

```
// External code
(function() {                    // isolated code
})();
// External code
```

**Note:** Placing part of the code between parenthesis transforms the function (structure) in an instruction; this is why a colon is needed at the end.