JAVASCRIPT

Lección 6: Functions (Funciones)

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Funciones

PRESENTACIÓN Y OBJETIVOS

En esta unidad se estudiarán las funciones, una parte fundamental de JavaScript. Vamos a conocer los diferentes tipos de funciones y las diferencias entre ellas.

Objetivos

En esta unidad aprenderás:

- ✓ **Qué es una función y su sintaxis**
- ✓ **Qué es una función anónima**
- ✓ **Qué son las function expressions**
- ✓ **Qué son las arrow functions**

1 FUNCIONES

Son una de las cosas fundamentales de JavaScript. Es un conjunto de instrucciones diseñadas para ejecutar una tarea en particular.

Una función es como un subprograma que puede ser llamado por código externo o interno (si se trata de recursividad).

En JavaScript las funciones son muy importantes. Son consideradas objetos de primera clase. Tienen propiedades y métodos igual que cualquier otro objeto. Son objetos Function.

El operando typeof devuelve en este caso 'function'.

```
typeof function myFunction() { console.log('a')} // function
```

1.1 Sintaxis

Una función se define usando:

- La palabra clave **function**
- El nombre de la función
  - Para el nombre de la función se aplican las mismas reglas que para el nombre de una variable (solo puede contener letras, números, _ y $ y no puede empezar por una letra)
- Una lista de los parámetros de la función, dentro de paréntesis () y separados por comas ,
- Las instrucciones que tienen que ejecutarse, dentro de llaves {}
  - Esta es la parte que se llama *function body* (el cuerpo de la función)

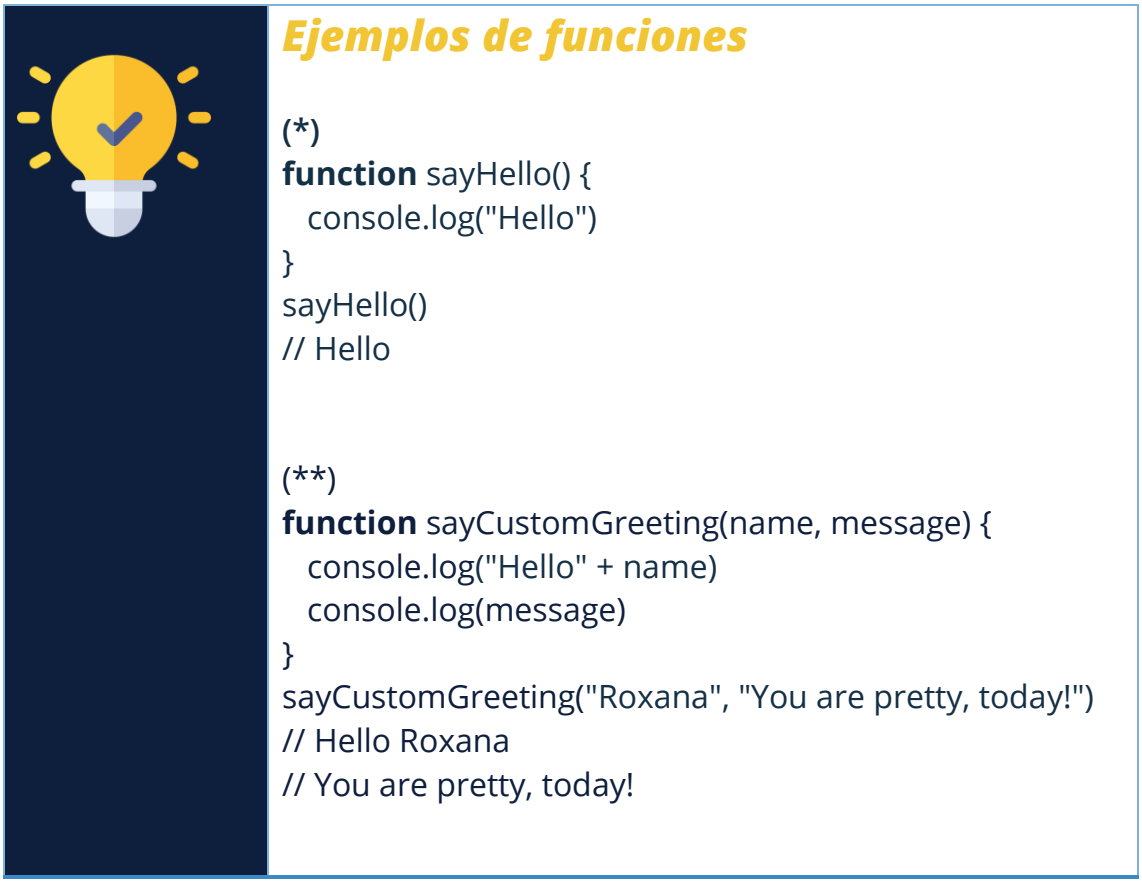
Declaración de una función

function functionName (**parámetro1, ... , parámetroN) {**
// function body (código a ejecutar)
}

functionName(valor1, ... valorN)

Para que una función se ejecute, no es suficiente con declararla. Hay que llamarla/invocarla. Para llamar a una función, hay que escribir su nombre seguido por paréntesis ().

Si la función tiene parámetros, dentro de los paréntesis hay que incluir valores para esos parámetros.



Ejemplos de funciones

```
(*)
function sayHello() {
  console.log("Hello")
}
sayHello()
// Hello


(**)
function sayCustomGreeting(name, message) {
  console.log("Hello" + name)
  console.log(message)
}
sayCustomGreeting("Roxana", "You are pretty, today!")
// Hello Roxana
// You are pretty, today!
```

(*) La primera función declarada se llama sayHello y no lleva ningún parámetro.

Se llama simplemente con el nombre sayHello, seguido por paréntesis (). La consola muestrera Hello.

(**) La segunda función se llama sayCustomGreeting y lleva 2 parámetros: nombre y mensaje. A llamarla le estamos pasando valores para esos parámetros. Es muy importante el orden de pasar los valores. El primer valor pasado corresponde al primer parámetro y el segundo valor corresponde al segundo parámetro.

Si una función espera 2 parámetros y al llamarla se le pasa sólo 1 valor, la función interpreta ese valor como el valor del primer parámetro y el valor del segundo parámetro como undefined.

Al contrario, si a una función al llamarla se le pasan más valores que parámetros, los valores extras se ignoran. No ocurre ningún error en la consola.

Los parámetros de una función no están limitados a cadenas de texto y números.

Hay funciones que al final de la ejecución devuelven algo (habitualmente un resultado computado en el cuerpo de la función).

Ejemplo de función que devuelve un valor

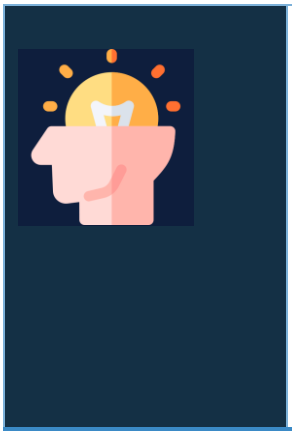
```
function calculateSquareArea(side) {
  console.log("I'm calculating the area...")
  return side * side
}

calculateSquareArea(5)
// I'm calculating the area ....

const squareArea = calculateSquareArea(3)
console.log("The area is " + squareArea)
// The area is 9
```

En el caso de la función que nos calcula el área de un cuadrado, podemos ver que al final del cuerpo de la función está la palabra **return** seguida por una computación.

return es la palabra clave que se usa para que la función nos devuelva algún valor. De esta forma, el valor será retornado a quien hizo la llamada.

Cuando hay un return, el resultado de la función se puede guardar en una variable.

En el momento que la función llega a la instrucción de return, detiene su ejecución. Cualquier instrucción después de return nunca llega a ejecutarse.



¡Presta atención!

```
function calculateSquareArea(side) {
  console.log("I'm calculating the area...")
  return side * side
  console.log("This will never be executed")
  console.log("Nor will this", side + side)
}
```

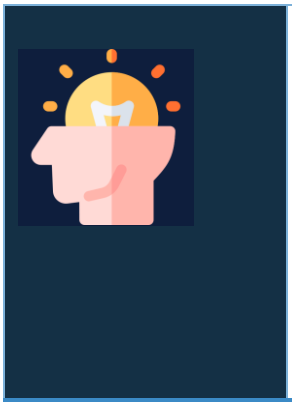
Las instrucciones marcadas con color rojo, siendo escritas después del return, nunca se van a ejecutar. La ejecución de la función se detiene después de return side * side.

¡Las funciones son cool! ☺

- Te permiten reutilizar código
- Defines el código una sola vez y lo usas varias veces
- Te permiten agrupar tu código y hacerlo más legible
- Puedes usar el mismo código con diferentes argumentos y obtener diferentes resultados

1.2 Ámbito de una función

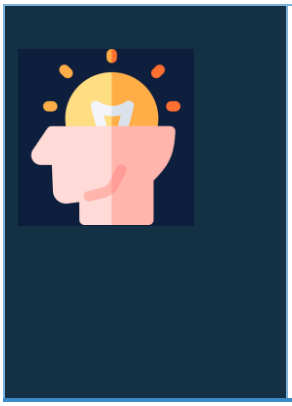
Las variables declaradas dentro de una función no pueden ser accedidas desde fuera de la función.



```
//  en este código no se conoce a la variable greeting

function hello() {
  var greeting = "Hello"
  console.log(greeting)
}
// en este codigo tampoco se conoce a la variable greeting
```

Las variables son locales para la función. Por esta razón, el mismo nombre de una variable se puede reutilizar en otra función. Función “a” puede tener dentro una variable “a”, y función “b” también puede declarar dentro una variable “a” sin causar ningún problema o error.

Una función tiene acceso a todas las variables definidas dentro del mismo ámbito que ella. Eso quiere decir que, si la función está declarada en el ámbito global, puede acceder a cualquier variable del ámbito global. De la misma manera, una función declarada dentro de otra función tiene acceso a todas las variables declaradas dentro de la función padre.



```
const greeting = "Hello"

function sayHello() {
  console.log(greeting)
}

sayHello() // Hello
```

En el ejemplo anterior, greeting está declarada en el ámbito global. De la misma forma que la función sayHello. Porque las dos están en el mismo ámbito, dentro de la funcion sayHello() podemos acceder a la variable greeting.

2 ANONYMOUS FUNCTIONS (FUNCIONES ANÓNIMAS)

Son funciones que no tienen un nombre asociado. Normalmente, cuando definimos una funciona, escribimos function functionName. En caso de las funcionen anónimas, el nombre se omite.

Una función anónima no es accesible después de su creación. Para poder reutilizarla, hay que guardarla como valor de una variable.

Podemos usar una función anónima para pasarla como argumento a otra función.

Sintaxis función anónima

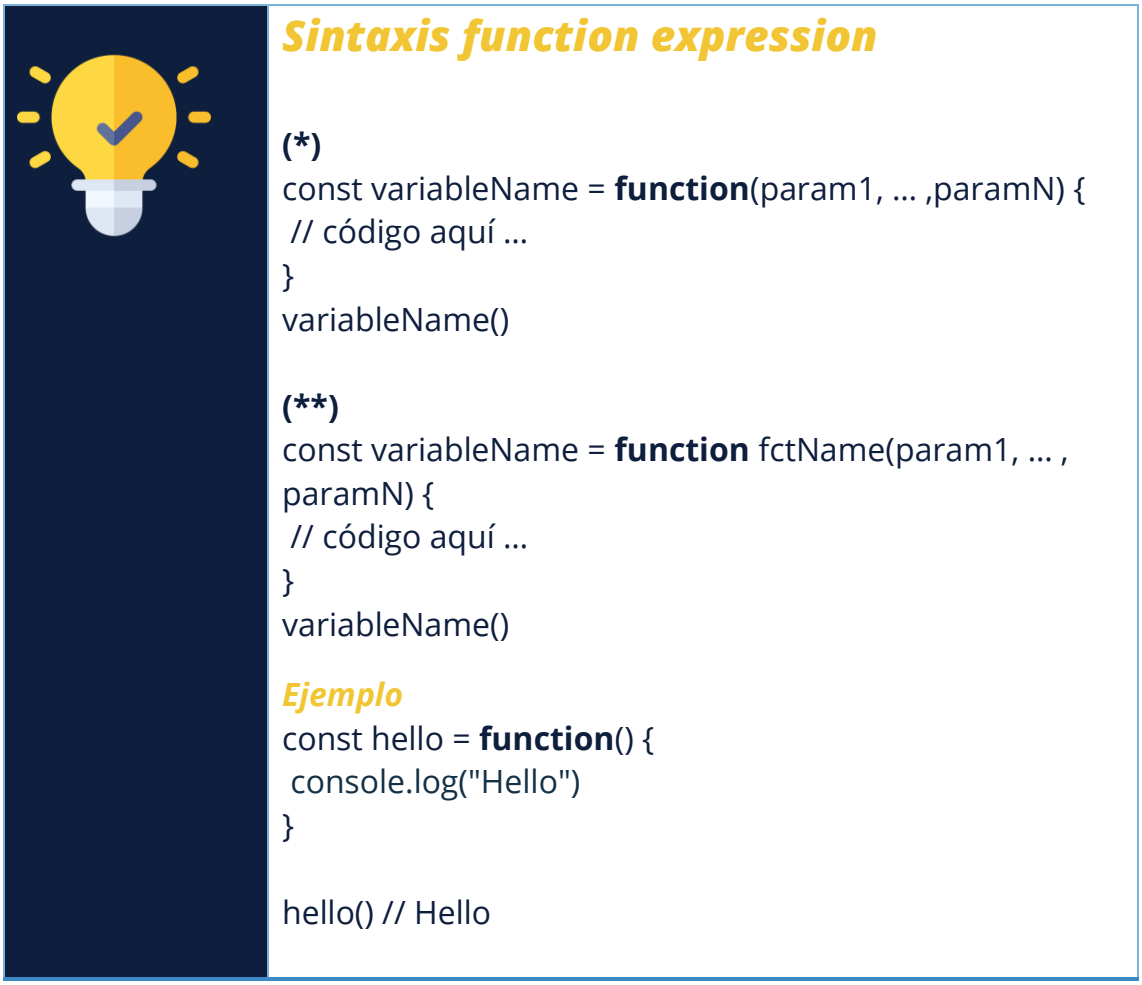
```
function() {
  console.log("This is an anonymous function")
}
```

Ejemplo pasando función anónima como parámetro a otra función

```
function doSomething(fct) {
  console.log("I will run the function that I get as a
parameter")
  fct()
}

doSomething(function() {
   console.log("I am an anonymous function")
})
//  I will run the function that I got as  a   parameter
//  I am an annonymus function
```

3 FUNCTION EXPRESSIONS

El concepto de function expression se refiere a crear una función y asignarla a una variable. La función puede ser anónima o no. Se suele usar con funciones anónimas.



Sintaxis function expression

(*)

```
const variableName = function(param1, ... ,paramN) {
 // código aquí ...
}
variableName()
```

(**)

```
const variableName = function fctName(param1, ... ,
paramN) {
 // código aquí ...
}
variableName()
```

Ejemplo

```
const hello = function() {
 console.log("Hello")
}

hello() // Hello
```

Como podemos ver, si se le pasa un nombre a la función (en vez de usar una función anónima), ese nombre no sirve de nada. La función no puede ser llamada por ese nombre. El único nombre que importa es el que se le da a la variable que tiene como valor la función.

Una de las principales diferencias entre functions y function expressions es que las funciones son hoisted y las function expressions no. En las lecciones sobre JavaScript avanzado, hablaremos más sobre este tema.

4 Arrow Functions

Las arrow functions fueron introducidas en ES6. Nos permiten escribir funciones de una manera mucho más corta.

Sintaxis arrow function

(*)
varName **= (**param1, ..., paramN**) =>** {
// código aquí
}

(**)
varName **= (**param1, ..., paramN**) => resultado**

En el caso de que la función retorne directamente un valor, no hace falta usar las llaves. Pero si el cuerpo de la función necesita más líneas de código, hay que usar las llaves y la palabra clave *return*.

Function Expression

```
const sayHello = function() {
    return "Hello World"
}
```

Arrow Function

```
const sayHello = () => "Hello World"
```

Las arrow functions pueden ser usadas como propiedades de un objeto, de la misma manera que las funciones normales.

5 PUNTOS CLAVE

- Las funciones son una parte fundamental de JavaScript
- Nos permiten agrupar código en bloques y reutilizar código
- La misma función puede ser llamada varias veces con varios parámetros
- Pueden tener parámetros y pueden retornar valores
- Para retornar algo de una función se usa la palabra clave return
- Todo lo que está escrito después de return, nunca llega a ser ejecutado
- Los objetos pueden tener como propiedades funciones
- Una función se puede asignar a una variable
- Las funciones anónimas son funciones sin nombre
- Arrow functions fueron introducidas en ES6 y nos permiten escribir funciones de formas más fácil