Universitat Oberta de Catalunya

Aula

M4.250 - Introducción a la programación en JavaScript I aula 1

Funciones

Inicio: **20/02/20**

Fin: **31/03/20**

Introducción

En este apartado trabajaremos el capítulo Funciones de la guía de Mozilla.

Podemos definir las funciones como trozos de código a los que ponemos un nombre para que nos sea más fácil reutilizarlos. Evidentemente es más complicado y, sobre todo, más completo que eso, pero como idea básica nos puede servir.

Aunque no nos lo parezca, en la práctica ya hemos usado alguna función. Cuando usamos console.log (loquesea), lo que estamos haciendo es llamar a una función que se encarga de escribir lo que toque en la consola. Así pues ya sabemos bastantes cosas de las funciones, que tienen un nombre, que tienen código dentro y que por llamarlas utilizaremos su nombre. Fácil, ¿no?

Definiendo las funciones

Podemos crear las funciones de maneras diferentes. Hay que tener presente que el resultado final será el mismo, pero dependiendo de lo que queramos hacer nos puede ser más práctico una manera u otra.

Declaración de función

La manera más parecida a la de otros lenguajes es utilizando la palabra clave **function** , seguido del nombre, los parámetros y el código que tiene esta función.

Los parámetros son unas variables / valores que podemos utilizar para que la función los use en su interior. Así si escribimos console.log ("Hola"); el parámetro será "Hola". Dentro de la función este valor se guardará en una variable. Cuando veamos nuestra primera función lo entenderemos mejor.

Definamos una función:

```
function suma (valor1, valor2) {
    return valor1 + valor2;
}
```

Desgranemos-la:

- **function** es la palabra clave que utilizaremos para definir la función.
- **suma** es el nombre de la función. Para crear nombres de funciones debemos tener en cuenta las mismas limitaciones y reglas que para crear nombres de variables.
- **valor1** y **valor2** son los parámetros. En la definición de la función son dos variables.
- **return** es una palabra clave que indica que esta función devolverá un valor. No siempre será así y podremos tener funciones que no devuelven ningún valor. En estos casos, no existirá una instrucción **return** .

Para llamar a esta función podemos hacer algo así:

```
let resultado = suma (3,2);
```

A **resultado** se guardará el valor que devuelva la función. Qué, como hemos visto, será la suma de 3 más 2.

Expresiones de funciones

Una segunda manera de crear una función es asignando a una variable:

```
let suma = function (valor1, valor2) {
    return valor1 + valor2;
}
```

En este caso hemos definido una función sin nombre y la hemos asignado a una variable. El resultado final es prácticamente el mismo.

La ventaja fundamental de este sistema es que podemos pasar estas funciones como parámetros de otras funciones. De momento no será algo que hagamos, así que no te preocupes demasiado. Pero es bueno tenerlo presente.

Llamando a las funciones

Ya hemos visto como cuando llamamos a una función, ponemos su nombre y los parámetros que sean necesarios. Pero debemos tener en cuenta algunas características:

- **Definición! = Ejecución** . Una función no se ejecuta si no se llama. Por lo tanto, aunque tengamos el código, no hará nada hasta que se llame.
- Por organización, a menudo ponemos la definición de las funciones al final del programa. Pero ¡cuidado! esto sólo funciona cuando utilizamos la declaración de funciones.
- Por lo tanto, si usamos una expresión de función la tendremos que poner antes de llamar a la función. Si no nos dará un error.

- Una función puede llamarse a sí misma (dentro del código de la función). Las funciones que se llaman a sí mismas se llaman funciones recursivas y a pesar de ser muy interesantes, hay que entenderlas muy bien antes de usarlas.

Ámbito de la función

Una de las gracias de las funciones es la posibilidad de aislar el código de la función del resto del programa. En este sentido, las variables definidas dentro de una función no pueden ser modificadas desde fuera. Y, aún más, podemos utilizar nombres de variables que ya estén definidas fuera de la función pero sin afectar al contenido de estas. Es más fácil con un ejemplo:

```
let a = 1, b = 2;
let c = suma (a, b);
console.log (a);
function suma (valor1, valor2) {
    let a;
    a = valor1 + valor2;
    return a;
}
```

¿Qué devolverá este programa? ¿1 o 3? Fácil 1. De hecho, en este programa podemos considerar que tenemos 6 variables:

1. a
2. b
3. c
4. suma.valor1
5. suma.valor2
6. suma.a

Las variables que están definidas dentro de la función tienen sentido dentro de la función. Desde fuera no se pueden llamar y no modifican en las que hay fuera.

Ahora bien, sí puede ocurrir que una variable definida fuera de la función sea modificada dentro:

```
let a = 1, b = 2;
let c = suma (a, b);
console.log (a);
function suma (valor1, valor2) {
    a = valor1 + valor2;
    return a;
}
```

En este caso, el valor que escribirá console.log será 3. ¿Por que? ¿qué diferencia hay con el código anterior? esta:

```
let a;
```

En este caso no hemos definido una variable dentro de la función. Por tanto, ha utilizado la variable que ya estaba definida fuera.

Es una buena práctica definir dentro de una función todas las variables que necesitamos. Y pasar como parámetros todos los valores que haya que utilizar en la función.

Esto no siempre es posible. Pero sólo tenemos que utilizar variables ya declaradas cuando nos interese que la función las modifique y siendo muy conscientes de que es así.

El objecto arguments

En JavaScript podemos crear funciones con un número de parámetros variables. El lenguaje nos proporciona una variable donde guarda todos los parámetros recibidos. De esta manera podemos acceder a todos los parámetros aunque no sepamos cuántos son.

arguments se comporta de una manera similar a como hemos accedido a los caracteres de un string. Así arguments[0] nos dará el primer valor que se haya pasado como parámetro. Y arguments.length nos dice el número de parámetros que la función ha recibido.

Funciones anidadas y cierresFuncions niuades i tancaments

Para resumir y simplificar. Podemos declarar funciones dentro de funciones. Se puede complicar cuando dentro de una función tenemos definida una función que a su vez tiene definida otra función. De nuevo, es un tema que vale la pena tener presente pero que no trabajaremos este curso.

Recursos de aprendizaje

Fuentes de información