POLITÉCNICO COLOMBIANO
JAIME ISAZA CADAVID

Educación para vivir mejor

Construcción de elementos de software web 1

Semana 9

JavaScript: Funciones

Media Técnica en Programación de Sistemas de Información
Politécnico Jaime Isaza Cadavid - 2020

¿Qué es una Función?

Una función es un bloque de código que **definimos una vez y lo reutilizamos las veces** que queramos; un conjunto de instrucciones a las que podemos pasar diferentes datos para que nos devuelva resultados distintos.

¿Para qué nos sirve crear funciones?

Las funciones son muy útiles a la hora **de crear un código único para usarlo en distintas partes de nuestro programa.** El beneficio de esto es que si en el futuro queremos modificar algo de ese código lo haremos **en un único sitio aunque se utilice en decenas de sitios diferentes.** Las funciones se ejecutan en distintos momentos y con distintas características gracias **a los *parámetros*.**

Otra de las ventajas de las funciones es que devuelven un valor, es decir, **realizan una operación y pueden devolver un dato.** Ese dato podemos asignarlo a una variable o usarlo dentro de otra operación. O incluso podemos prescindir de él si no nos interesa para nada.

Las funciones son una forma de **agrupar código que vamos a usar varias veces permitiéndonos además, pasar diferentes valores para obtener diferentes resultados.**

Podemos intentar hacer un paralelismo con el café, más o menos todo el mundo sabe hacer un café, desde los ingredientes a los diferentes pasos. Y cada vez que queramos uno seguimos todos los pasos uno a uno, y al final tendremos un café.

Ahora, las funciones serían como estas cafeteras modernas, a las que dependiendo de la cápsula que uses te hace un café diferente. Cuando activo la cafetera ***(invoco a la función)*** detecta qué tipo de cápsula he introducido ***(parámetros)*** y me hace ***(me devuelve)*** un café u otro.

Un ejemplo de esto sería:

```
function hacerCafe( nombreDelCafe ) {
  return `Aquí tiene su ${nombreDelCafe}, que lo disfrute`;
}
```

De manera que podemos llamar varias veces a la función y obtener "cafés" diferentes:

```
hacerCafe( 'Café Juan Valdez' );
// devuelve "Aquí tiene su Café Juan Valdez, que lo disfrute"

makeMeCoffee( 'Café Colcafe' );
// devuelve "Aquí tiene su Café Colcafe, que lo disfrute"
```

¿En qué casos utilizamos las funciones?

Por ejemplo, en los siguientes casos:

- Si tenemos un código que convierte la primera letra de un texto a mayúsculas y vamos a usar ese código en varias partes de nuestro programa, creamos una función y ejecutamos la función en cada uno de los sitios necesarios.
- Si queremos añadir varias clases a diferentes elementos HTML en función de la medida de la página web podemos crear una función y utilizarla en cada uno de ellos.
- Si queremos enviar datos a un servidor, la mayoría de las veces es muy parecido y sólo cambian unos datos. Podríamos hacer una función y reutilizarla y usar distintos datos en cada una mediante los parámetros de la función.

Declaración y uso de las funciones

Para utilizar una función debemos declararla en algún sitio de nuestro código.

La estructura para declarar una función es

- primero la palabra reservada **function** seguida del **nombre de la función**
- después entre paréntesis **( )** los **parámetros** de la función separados por comas **,** y que si no tiene parámetros **estará vacío**.
- un bloque de código entre llaves **{ }** con las instrucciones de código de la función.

Veamos otro ejemplo teniendo en cuenta estos pasos para declarar una función:

```
//Función sin parámetros
function saludo() {
    return 'Hola';
}

//Función con parámetros
function suma(numeroUno, numeroDos) {
    return numeroUno + numeroDos;
}
```

Si añadimos las declaraciones anteriores de funciones a nuestra página, no veremos ningún efecto. **Esto es porque solo estamos declarando las funciones, es decir, diciendo que existen pero nada más.**

Para utilizar **(también se le puede llamar *ejecutar* o *invocar*)** una función simplemente usamos el **nombre de la función seguida de paréntesis donde pasaremos los parámetros o argumentos separados por comas , .** Como en el siguiente ejemplo:

```
console.log( saludo() );
//Muestra en la consola la palabra 'Hola'

console.log( suma(1, 4) );
//Muestra en la consola un 5
```

NOTA: *Esta sintaxis para utilizar funciones te suena, ¿verdad? Hasta ahora hemos estado ejecutando algunas funciones ya declaradas en el navegador como* ***querySelector('.title')*** *a la que le* ***pasamos por parámetro*** *una cadena con el selector que buscamos y nos devuelve la referencia a dicho elemento en nuestro HTML.*

Se pueden crear funciones sin nombre, estas funciones se llaman ***funciones anónimas***. Estas funciones se suelen emplear para aspectos y funcionalidades que se van a ver en temas posteriores, como asignarlas a una propiedad de un objeto o pasarlas como un callback. Un ejemplo de función anónima:

```
const sum = function(a, b) {
    return a + b;
};

// La llamamos con el nombre de la variable
sum(2,3); // devuelve 5
```

Parámetros y valores de retorno

Los ***parámetros*** son los datos que definimos en una función y que, a la hora de ejecutarla, serán sustituidos por los ***argumentos*** que le pasemos. Por tanto, **en la declaración de la función le llamamos parámetros y en la ejecución le llamamos argumentos.** Las funciones pueden tener 0, 1 o más parámetros separados por comas **,** .

Una función puede **devolver un valor** utilizando la palabra clave **return** seguida del valor que queremos devolver. Para devolver una variable **result**, utilizaremos **return result;** en el código.

```
function sum(a, b) {
    const result = a + b;

    return result;
}

const sumResult = sum(3, 4); //sumResult vale 7
```

Por defecto, si en una función **no indicamos un valor de retorno** usando **return,** la función devolverá el valor **undefined**. El valor ***undefined*** en JavaScript **indica que una variable ha sido declarada pero no posee ningún valor,** en este caso determina que la función no tiene **asignado ningún valor de retorno y por eso devuelve undefined.**

Cuando ejecutamos una instrucción **return** dentro de una función, termina la ejecución de la función. **Todo el código que se fuese a ejecutar después de ese return será ignorado,** como si no existiese. Por tanto, debemos **evitar escribir líneas de código después de un return** y **normalmente será la última línea de código de una función.**

Ámbito de las variables

Por defecto, una variable definida con **let o cons**t tiene un **ámbito (en inglés, *scope*)** que corresponde a su bloque, es decir, van a existir dentro de su bloque.

¿Y qué es un bloque? **Un bloque es cualquier expresión con llaves {} como puede ser un condicional, un ciclo o una función.**

Por ejemplo:

```
const variableGlobal = 'Ey!, Soy global';

// ejemplo para asegurarnos de que entra en el bloque if
if (2 === 2) {
  const variableGlobal = 'Ey!, Realmente no soy Global';
  const variableNoGlobal = 'No soy global :(';

  // devuelve "// devuelve "Ey!, Realmente no soy Global"
  console.log( variableGlobal  );
    // devuelve "No soy global :("
  console.log( variableNoGlobal );
}
// devuelve "Ey!, Soy global"
console.log( variableGlobal );
// da un error porque no está definida
console.log( variableNoGlobal );
```

Por supuesto, podemos acceder a las variables del ámbito superior:

```
let globalVar = 'Ey, I\'m global';
if (2 === 2) {
  globalVar = 'Ey, I\'m STILL global';
  // devuelve "Ey, I'm STILL global"
  console.log( globalVar );

}
// devuelve "Ey, I'm STILL global" que se cambió en el bloque if
console.log( globalVar );
```

De esta manera, una variable creada dentro del cuerpo de una función **sólo será accesible desde dentro de esa función.**

Desde dentro de una función podemos **utilizar las variables que se hayan definido fuera de cualquier función**, y **gracias al ámbito de cada función también podemos crear, sin generar conflicto, nuevas variables que se llamen como variables de otras funciones.**

Por ejemplo:

```
function f1() {
    const item = 1;
    return item;
  }

  function f2() {
    const item = 2;
    return item;
  }

  console.log( f2() ); // devuelve 2;
  console.log( f1() ); // devuelve 1;
```

Funciones flecha - Arrow Functions

Las arrow functions ("funciones flecha") de ES6 son una forma simplificada para declarar funciones anónimas. La sintaxis básica es la siguiente:

```
const sum = (a,b) => {
    return a + b;
 };

 // y la ejecutamos usando la variable a la que la hemos asignado:
 sum(2,3); // devuelve 5;

 // Anteriormente vimos esta misma función con la forma "normal"
 const sum = function(a,b) {
   return a + b;
 };
```

En las funciones flecha podemos evitar los paréntesis solo cuando la función tenga 1 único parámetro:

```
const printWaitingTime = minutes => {
    console.log(`Please, wait ${minutes} minutes`);
 };

 // equivale a
 const printWaitingTime = (minutes) => {
   console.log(`Please, wait ${minutes} minutes`);
 };
```

Escribir o no las llaves **({})** significa dos cosas distintas. **Solo podremos no escribirlas cuando la función tenga una sola sentencia; es decir, cuando se ejecute una sola orden** dentro (un console.log(), un cambio en un elemento HTML, un incremento en un contador, etc.). Cuando no escribimos las llaves, el valor que devuelve esa sentencia será el r**eturn de la función.** Eso nos permite escribir en menos líneas funciones muy sencillas:

```
const printWaitingTime = minutes => `Please, wait ${minutes} minutes`;
console.log( printWaitingTime(4) );
// devuelve "Please, wait 4 minutes"

// equivale a
const printWaitingTime = (minutes) => {
  return `Please, wait ${minutes} minutes`;
};
console.log( printWaitingTime(4) );
// devuelve "Please, wait 4 minutes"
```

Recursos externos

Curso de Ada Lovelace en Youtube

Este curso explica bastante bien y de forma breve qué son las funciones y cómo trabajar con ellas.

- Funciones. Introducción
- Funciones anónimas
- Funciones. Parámetros y argumentos
- Ámbito de variables

Libros Web

Si prefieres un recurso escrito para aprender, aquí tienes la explicación de las funciones y el ámbito de las variables en JavaScript de la mano del libro de Introducción a JavaScript por Libros Web.

- Funciones
- Ámbito de las variables

Ejercicio para abordar en clase con los estudiantes

EJERCICIO Cálculo de administración de inmuebles

Teniendo en cuenta los conocimientos previos de los estudiantes en el grado décimo de fundamentos de programación en el módulo de Desarrollo del Pensamiento Analítico y Sistémico. Se debe realizar el ejercicio adjunto a esta semana donde se logre evidenciar los conceptos claves de estructuras secuenciales, condicionales, repetitivas, funciones y en la medida de lo posible vectores y matrices.

Taller

EJERCICIO 1

Función multiplicación

Crea una función que reciba como argumento dos valores y devuelva como valor de retorno la multiplicación de ambos. Haz tres pruebas con distintos números para comprobar que funciona correctamente y muestra el resultado en la consola usando console.log().

EJERCICIO 2

Función media

Crea una función que recibe 4 parámetros, cada uno con un número, y devuelva como valor la media de todos ellos. Haz tres pruebas con distintos números para comprobar que funciona correctamente y muestra el resultado en la consola.

EJERCICIO 3

Ticket con IVA

Crea una función que reciba como parámetro un número, que representará un precio, y devuelva un texto en el que ponga el precio sin IVA, el IVA (21%) y el total. Por ejemplo, si pasamos por parámetro un 10, la función pintará en la consola "Precio sin IVA: 10, IVA: 2,1 y Total: 12,1".

Para probar que funciona, ejecuta la función recogiendo el resultado en una variable e imprímela en la consola para comprobarlo.

EJERCICIO 4

Calculador de modelo de caja

Como hemos visto en las clases anteriores, en CSS tenemos dos tipos de cálculo para las dimensiones de un elemento: border-box y content-box. Vamos a realizar un calculador al que le pasaremos 4 parámetros y nos devolverá el ancho del contenido, en caso de ser *border-box* o el ancho total de la caja, en caso de ser *content-box*.

La función tendrá 4 parámetros: el primero será un booleano para especificar si es border-box o no, el segundo será el ancho del contenido o de la caja entera, el tercero el padding y el cuarto el borde.

Para probar que funciona, ejecuta la función recogiendo el resultado en una variable e imprímela en la consola para comprobarlo.