MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO DE WEB ENTORNO CLIENTE
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Funciones y objetos

07

ÍNDICE

OBJETIVOS

Saber crear funciones propias.

Conocer las principales características de las funciones.

Saber crear y utilizar objetos mediante clases.

Comprender los mecanismos de invocación de funciones.

Conocer los objetos nativos del lenguaje y sus funciones.

/ 1. Introducción y contextualización práctica

JavaScript es un lenguaje muy potente que ofrece un gran número de funcionalidades en lo que se refiere a las funciones y a los objetos disponibles en el lenguaje. La biblioteca de JavaScript ofrece un número bastante grande de funciones nativas que se ponen a disposición del programador.

A las funcionalidades ya existentes en JavaScript hay que sumar aquellas que existen en otras librerías, ya que permiten aumentar el número de funcionalidades disponible de manera significativa. Como veremos en el tema, la utilización de funciones en cualquier lenguaje de programación y, por lo tanto, en JavaScript, permite tener una mejor modularidad del código y facilitar el mantenimiento futuro.

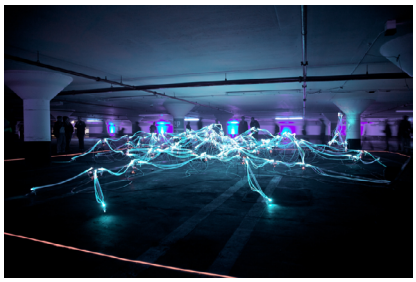

Fig. 1. La definición de funciones potencia nuestro código

De forma general, cuando observemos que tenemos un gran número de sentencias que se repiten a lo largo de todo el código, es recomendable refactorizar. Es decir, crear una función que agrupe todas estas sentencias y permita su reutilización en cualquier parte del código.

Escucha el siguiente audio que describe el caso práctico que iremos resolviendo a lo largo de la unidad:

Audio Intro. Elección de tipo de funciones en JavaScript
https://bit.ly/2URSiLM

/ 2. Definición de funciones

Podríamos definir una función como un conjunto de instrucciones que realizan una tarea determinada.

Cuando se desarrolla una aplicación, ya sea en un entorno web o no, casi siempre es una buena práctica estructurar el código por módulos. De esta manera, garantizamos que se pueda reutilizar el código a lo largo de nuestra solución, evitando tener código duplicado. Por la forma de programar que tenemos los seres humanos, estamos acostumbrados a repetir una y otra vez las mismas instrucciones para solucionar un mismo problema. En este sentido, es fácil observar que si las instrucciones se repiten una y otra vez, crearemos redundancia y complicaremos, en gran medida, el código.

El uso de las funciones nos permite estructurar el código de una manera sencilla y evitar, así, duplicar instrucciones. JavaScript, al igual que otros lenguajes de programación, permite la declaración de funciones a lo largo del código.

Desde el punto vista informático, hay una similitud entre una función algorítmica y una función matemática. Por ejemplo, una función matemática recibe un parámetro y genera un resultado. Algo similar ocurre con una función en código. Veamos un ejemplo:

```
function exampleFunction(a, b) {
 return a * b;  //La función devuelve el producto de los parámetros
}
```

Código 1. Función en código

Como se puede observar, organizando las instrucciones en una función, se pueden generalizar y evitar la redundancia. De esta forma, el código es más fácil tanto de mantener, como lo será realizar modificaciones y extensiones en el futuro.

Vídeo 1. Reutilización de código con funciones JavaScript
https://bit.ly/2zAlKOL

2.1. Sintaxis para la creación de funciones

Una función en JavaScript requiere de la utilización de la palabra reservada function. A continuación, se incluye un identificador con el nombre de la función. Nótese que las reglas para crear este identificador son exactamente iguales que las reglas para definir los identificadores de las variables. En este sentido, tras el nombre de la función, se incluye la lista de argumentos entre paréntesis. Una función puede tener 'n' argumentos, siendo n mayor o igual que cero.

Los argumentos de una función consisten en una lista de valores necesarios para que esta pueda ejecutar el bloque de código correspondiente.

Los nombres de los argumentos dentro de la función se comportan como variables, exactamente igual que en un bloque fuera de una función. Se dice que el ámbito de visibilidad de una variable dentro de una función es el cuerpo exacto de la función. Es decir, fuera de una función esa variable no va a existir, al igual que sucede con sus parámetros. Por ello es posible utilizar el mismo nombre de variables en otras funciones o en el código principal de nuestra solución.

En otros lenguajes de programación se diferencia entre procedimiento/subrutina y funciones.

La principal diferencia entre estos reside en la devolución del resultado, pues una función devuelve un resultado.

Una subrutina, por su parte, no devuelve nada. JavaScript no diferencia entre la definición de subrutinas y funciones: ambas se definen con la palabra reservada function. Un ejemplo de procedimiento podría ser el siguiente:

```
function procedureExample(a, b) {
  console.log("El valor es: "+(a*b)); //No se devuelve nada
}
```

Código 2. Ejemplo de procedimiento que no devuelve nada

Audio 1. Sintaxis para la creación de funciones
https://bit.ly/2CcAMvb

2.2. Objeto Function

Otra forma de crear funciones en JavaScript es a través del constructor Function. A través de él se pueden especificar los parámetros y el cuerpo de la función como si fuesen literales. En la actualidad no es una solución ampliamente utilizada, pero es importante que sepamos que existe, porque puede haber mucho código legacy (código heredado).

```
var func = new Function("a", "b", "return a * b");
//Invocación de la función a través del objeto creado previamente
var resultado = func(10,10);
```

Código 3. Ejemplo del constructor Function a través de un literal

2.3. Invocación de funciones

Para poder ejecutar el código de una función es necesario indicar al lenguaje que nos referimos a ese fragmento de código. Para ello, se utiliza el mecanismo de invocación, que consiste en llamar a la función por su nombre para obtener un resultado o realizar un procesamiento de datos.

En JavaScript las funciones pueden ser invocadas a través de una sentencia, y se generará un resultado cuando finalice su ejecución. El resultado devuelto por la función se puede utilizar en el programa principal o en el flujo de ejecución de otra función. Como se puede observar, las funciones pueden invocarse dentro de otras, es decir, las llamadas se pueden anidar. Una característica de JavaScript es que las funciones se pueden utilizar exactamente igual que las variables, tanto en sentencias como en asignaciones, así como en el resto de las situaciones que nos ofrezca el código.

En JavaScript las funciones y las variables pueden ser utilizadas antes de su declaración. Existe un mecanismo por el cual JavaScript puede mover todas las declaraciones al principio del código con la idea de facilitar la lectura de éste por otros desarrolladores. Esto es conocido como hoisting.

```
ejemploFunc(5);
//Declaración de la función
function ejemploFunc(x) {
  return x*x;
}
```

Código 4. Función que se utiliza antes de su declaración

/ 3. Caso práctico 1: "Web para navegadores antiguos"

Planteamiento. Nuestro mayor cliente nos pide crear una web que incorpore ciertas funcionalidades en JavaScript. Uno de los requisitos más importantes es que la página pueda funcionar en navegadores muy antiguos. Nuestro equipo decide afrontar el problema mediante funciones para mejorar la modularidad del código y poder reutilizarlo en otras soluciones. Para la definición de funciones se decide utilizar el objeto Function y evitar pasar parámetros a las funciones (todo se hace con variables externas).

Nudo. ¿Qué opines de esta solución? ¿Se puede mejorar en algún punto?

Desenlace. Como hemos visto en el apartado sobre la definición de funciones, actualmente no es recomendable utilizar el constructor Function. Además, al definir las funciones como literales, no es posible detectar posibles errores sintácticos. El hecho de no pasar parámetros es una mala decisión, pues se utilizarán variables globales, lo que dificulta exactamente uno de los puntos que se buscan, la modularidad y reutilización/portabilidad.

/ 4. Definición de funciones (II)

4.1. Funciones autoinvocadas

En JavaScript existe un tipo especial de funciones que son denominadas funciones autoinvocadas. Este tipo de funciones se pueden invocar por sí mismas, es decir, sin que ningún fragmento de código realice dicha invocación.

```
(function () {
 var msg = "Hola!!";  // Esto se ejecutará sin llamar explícitamente
a la función})();
```

Código 5. Función autoinvocada

4.2. Funciones anónimas

Javascript ofrece la posibilidad de crear lo que se llaman funciones anónimas, es decir, funciones que no tienen un nombre definido. El mecanismo para poder crear estas funciones es asignar la función a una variable determinada. Desde ese momento se podrá utilizar esa variable como si fuese la propia función. El problema de esta aproximación es que se pierde visibilidad de lo que es una variable y lo que es una función, dificultando en cierta medida el desarrollo, sobre todo cuando nos estamos familiarizando con el lenguaje.

```
var func = function(a,b){return a *b};
//Invocación de la función anónima a través de la variable declarada
var result = func(10,5);
```

Código 6. Función anónima que se asigna a una variable

4.3. Funciones con notación flecha

Las funciones con notación flecha, conocidas como arrow functions, son una forma sintácticamente reducida de definir funciones. Se basan en la utilización de funciones anónimas y en la eliminación de las palabras reservadas para la definición de la función.

```
const arrowF = (a, b) => a * b;
```

Código 7. Función con notación flecha

Algunas características de estas funciones son las siguientes:

- No son adecuadas para definir métodos en los objetos.
- Deben ser declaradas antes de su uso, por lo que no se mueven al comienzo del código automáticamente (no realizan hoisting).
- En su declaración es mejor utilizarlas como constantes que como variable, puesto que la definición de la de la función va a ser siempre constante, es decir, no cambiarán con el tiempo.

4.4. Paso de parámetros

En JavaScript existen unas reglas para la definición y el paso de los parámetros de una función:

- Los tipos de los parámetros no se especifican durante su definición.
- No se comprueban los tipos en los argumentos pasados.
- No se comprueba el número de parámetros pasados a la función.

En JavaScript existe la posibilidad de definir parámetros por defecto para evitar errores en la llamada a funciones. Si un parámetro no se pasa a la función, normalmente se trata como indefinido, lo cual puede provocar errores de ejecución. En ocasiones se asume este riesgo, pero en muchas circunstancias es mejor asignar un valor por defecto a los parámetros para prevenir posibles errores.

En algunos lenguajes de programación el paso de parámetros puede ser por valor, por referencia o incluso por copia.

En JavaScript los parámetros de una función se pasan siempre por valor, lo que significa que la función solo toma el valor de los parámetros y no la localización de estos argumentos. Los cambios realizados sobre los argumentos no se transmiten fuera del cuerpo de la función.

Por el contrario, los objetos se pasan por referencia, por lo que los cambios que realicemos en un objeto dentro de una función se verán reflejado en el cuerpo de nuestro programa.

```
//Protección del parámetro a por si no es pasado por parámetro
function testFunc(a, b) {
 if (a === undefined) {
   a = 0;
 }
}

//Mejor aproximación para asignar valores por defecto a los parámetros
function testFunc(a=1, b=0) {
 ...
}
```

Código 8. Ejemplo de paso de parámetros

4.5. Funciones anidadas

En JavaScript todas las funciones tienen acceso al ámbito de visibilidad superior, al igual que sucede fuera de la función. Existe la posibilidad de crear funciones anidadas que, por definición, son funciones que se declaran en el interior de otra función. Las funciones anidadas tienen visibilidad justo hasta el nivel superior, es decir, hasta el cuerpo de la función que la contiene. Las funciones anidadas pueden complicar, en cierta medida, la comprensión del código, como sucede con las funciones anónimas. Una función anidada no puede ser invocada fuera de la función en la que fue definida.

```
function parentFunc() {
  var var1 = 0;
  function nestedFunc() {counter += 1;}
  nestedFunc();
  return var1;
}
```

Código 9. Función anidada

4.6. Funciones nativas del lenguaje

JavaScript proporciona un gran número de funciones nativas al propio lenguaje. Es importante tener en cuenta que estas funciones no deberían ser llamadas métodos. Sin embargo, si consideramos que todas ellas están declaradas en el objeto global podría tener sentido que, en algunos casos, se denominasen métodos.

En el lenguaje se proporcionan diferentes métodos para trabajar con diferentes tipos de datos. Por ejemplo, se pueden utilizar funciones para transformar cadenas de caracteres a representación numérica, es posible transformar objetos a cadenas y viceversa, se puede evaluar un código JavaScript a partir de una representación en cadena, etc.

Fig. 3.Funciones en JavaScript

/ 5. Objetos

En JavaScript se permite la definición de clases de objeto a través de la palabra reservada class. Se considera que una clase es un tipo especial de función que puede contener propiedades, así como un método para construir el objeto de dicha clase.

El constructor es un método especial para poder crear e inicializar los objetos que pertenecen a una clase que se invoca automáticamente. A diferencia de otros lenguajes, en JavaScript solo puede existir un constructor por clase.

Como JavaScript es un lenguaje orientado a objetos, se permite la herencia a través de la palabra reservada extends. Por medio de la herencia podemos crear una jerarquía de clases y reutilizar el código, en la medida de lo posible.

Fig. 4. Objetos en JavaScript

5.1. Objeto String

JavaScript ofrece esta clase para poder albergar cadenas de caracteres y proporcionar los métodos oportunos para tratarlos de una manera sencilla para el programador. Algunos de los métodos más relevantes de esta clase son los siguientes:

Método	Descripción
concat()	Permite unir dos cadenas de caracteres
match()	Busca una cadena de caracteres en otra
indexOf()	Devuelve la posición de una cadena dentro de otra
slice()	Permite obtener una subcadena
split()	Divide una cadena en un array de cadenas

Tabla 1. Métodos de la clase String

5.2. Objeto Math

Este objeto se utiliza para llevar a cabo operaciones matemáticas.

No requiere de un constructor y los métodos se pueden acceder de manera estática. Aunque esta clase dispone de un gran número de métodos, los más utilizados son el valor absoluto y el redondeo.

```
//Funciones de redondeo
Math.round(7.7);   // returns 8
Math.round(7.4);   // returns 7
```

Código 10. Uso del objeto Math

5.3. Objeto Date

Este objeto permite manejar fechas y tiempo con relativa sencillez y así libera al programador de una tarea que en ocasiones puede ser realmente tediosa. Algunas de las funciones más importantes de esta clase son las siguientes:

Método	Descripción
getDate()	Devuelve el día del mes (1-31)
getDay()	Proporciona el día de la semana (0-6)
getTime()	Devuelve el número entre milisegundos desde el 1 de enero de 1970 y otra fecha
now()	Devuelve el número de milisegundos desde 1970
parse()	Convierte una cadena con una fecha a un objeto

Tabla 2. Métodos de la clase Date

Vídeo 2. "Trabajando con los objetos nativos"
https://bit.ly/2YHILaW

/ 6. Caso práctico 2: "Definición y visibilidad de funciones"

Planteamiento. Una de las funcionalidades que ofrece JavaScript es poder crear funciones e invocarlas allí donde sean visibles. Para dar solución a un problema que produce un error en una web de un cliente, nuestro experto en JavaScript nos dice que el error que se genera en la web se da debido a que las funciones están declaradas después de su utilización.

Nudo. ¿Crees que el experto tiene razón?

Desenlace. Como hemos visto en el apartado sobre la invocación de funciones, en JavaScript es posible realizar la invocación antes de su declaración formal. Esto es gracias a que JavaScript permite realizar hoisting de las funciones automáticamente. Lo que sí es cierto es que las funciones que se declaran con la notación flecha no pueden ser invocadas sin declararse previamente, pues el lenguaje no las puede desplazar al comienzo de forma automática.

Fig.5. La definición de funciones y su visibilidad es fundamental

/ 7. Resumen y resolución del caso práctico de la unidad

A largo de este tema hemos presentado diferentes formas de trabajar con las funciones, así como las diferentes maneras de crearlas en JavaScript. Es importante conocer cada una de las formas de crear funciones, pues existen en la actualidad un gran número de librerías que las crean y manejan de diferentes formas, atendiendo a todos los mecanismos que ofrece JavaScript.

Ha sido interesante también comprender el funcionamiento que tienen las funciones anónimas y la creación de objetos mediante este tipo de funciones ya que son algunas de las tendencias más utilizadas en las soluciones reales. Aunque existen algunos mecanismos de creación de funciones que no se utilizan tanto, es importante tenerlos en cuenta por si nos surge la necesidad de utilizarlas en cualquier problema.

Además, a lo largo del tema hemos presentado diferentes mecanismos para la creación y manipulación de objetos propios y nativos del lenguaje, al igual que las diferentes funciones nativas del lenguaje. Es recomendable revisar todo el API de JavaScript antes de crear una nueva función, pues como ya hemos dicho, JavaScript ofrece un gran número de funcionalidades ya predefinidas.

Resolución del caso práctico de la unidad

Como hemos visto a lo largo del tema, JavaScript ofrece diferentes mecanismos para la creación de funciones, siendo uno de ellos la 'notación flecha'.

Se trata de un tipo de función que puede inducir a error al ser declaradas inline. Además, estas funciones deben ser declaradas siempre antes de su utilización, limitando en gran medida la potencia del lenguaje.

Al utilizar funciones con notación flecha no podremos crear objetos, pues este tipo de funciones no permiten definir métodos.

Por todo esto, el enfoque planteado inicialmente no es adecuado. Una alternativa sería establecer una política más flexible que contemple qué sucede con los objetos y que limite, en la medida de lo posible, las funciones tipo flecha.

/ 8. Bibliografía

Flanagan, D. (2020). JavaScript: *The Definitive Guide. (7th edition)*. Sebastopol: O'Reilly Media, Inc.

Crane, D., McCarthy, P. y Tiwari, S. (2008). *Comet and Reverse Ajax: the next-generation Ajax 2.0.* Berkeley: Springer-Verlag.

Dean, J. (2019). *Web programming with HTML5, CSS, and JavaScript.* Burlington: Jones & Bartlett Learning.