Funciones de JavaScript

Las funciones de JavaScript se **definen** con la functionpalabra clave.

Puede utilizar una **declaración** de función o una **expresión** de función .

Declaraciones de funciones

Anteriormente en este tutorial, aprendió que las funciones se **declaran** con la siguiente sintaxis:

```
function functionName(parameters) {
 // code to be executed
}
```

Las funciones declaradas no se ejecutan inmediatamente. Se guardan para su uso posterior y se ejecutarán posteriormente, cuando se invoquen.

Ejemplo

```
function myFunction(a, b) {
 return a * b;
}
```

El punto y coma se utiliza para separar sentencias ejecutables de JavaScript.
Dado que una **declaración** de función no es una sentencia ejecutable, no es habitual terminarla con punto y coma.

Expresiones de función

Una función de JavaScript también se puede definir mediante una **expresión** .

Una expresión de función se puede almacenar en una variable:

Ejemplo

```
const x = function (a, b) {return a * b};
```

Después de que una expresión de función se haya almacenado en una variable, la variable se puede utilizar como una función:

Ejemplo

```
const x = function (a, b) {return a * b};
let z = x(4, 3);
```

La función anterior es en realidad una **función anónima** (una función sin nombre).

Las funciones almacenadas en variables no necesitan nombre de función. Siempre se invocan (llaman) usando el nombre de la variable.

La función anterior termina con un punto y coma porque es parte de una declaración ejecutable.

El constructor Function()

Como has visto en los ejemplos anteriores, las funciones de JavaScript se definen con la functionpalabra clave.

Las funciones también se pueden definir con un constructor de funciones de JavaScript integrado llamado Function().

Ejemplo

```
const myFunction = new Function("a", "b", "return a * b");

let x = myFunction(4, 3);
```

En realidad, no es necesario usar el constructor de funciones. El ejemplo anterior es el mismo que escribir:

Ejemplo

```
const myFunction = function (a, b) {return a * b};

let x = myFunction(4, 3);
```

La mayoría de las veces, puedes evitar el uso de la newpalabra clave en JavaScript.

Función de elevación

Anteriormente en este tutorial, aprendiste acerca del "elevación" (JavaScript Hoisting).

La elevación es el comportamiento predeterminado de JavaScript para mover **declaraciones** a la parte superior del ámbito actual.

La elevación se aplica a las declaraciones de variables y a las declaraciones de funciones.

Gracias a esto, las funciones de JavaScript se pueden llamar antes de ser declaradas:

```
myFunction(5);

function myFunction(y) {
 return y * y;
}
```

Las funciones definidas mediante una expresión no se elevan.

Funciones autoinvocables

Las expresiones de función pueden hacerse "autoinvocables".

Una expresión autoinvocada se invoca (inicia) automáticamente, sin ser llamada.

Las expresiones de función se ejecutarán automáticamente si la expresión es seguida por ().

No es posible autoinvocar una declaración de función.

Debes agregar paréntesis alrededor de la función para indicar que es una expresión de función:

Ejemplo

```
(function () {
 let x = "Hello!!"; // I will invoke myself
})();
```

La función anterior es en realidad una **función anónima que se invoca a sí misma** (función sin nombre).

Las funciones se pueden utilizar como valores

Las funciones de JavaScript se pueden utilizar como valores:

Ejemplo

```
function myFunction(a, b) {
 return a * b;
}

let x = myFunction(4, 3);
```

Las funciones de JavaScript se pueden utilizar en expresiones:

Ejemplo

```
function myFunction(a, b) {
 return a * b;
}

let x = myFunction(4, 3) * 2;
```

Las funciones son objetos

El typeofoperador en JavaScript devuelve "función" para funciones.

Pero las funciones de JavaScript se pueden describir mejor como objetos.

Las funciones de JavaScript tienen **propiedades** y **métodos** .

La arguments.lengthpropiedad devuelve el número de argumentos recibidos cuando se invocó la función:

Ejemplo

```
function myFunction(a, b) {
 return arguments.length;
}
```

El toString()método devuelve la función como una cadena:

Ejemplo

```
function myFunction(a, b) {
 return a * b;
}

let text = myFunction.toString();
```

Una función definida como propiedad de un objeto se denomina método del objeto.
Una función diseñada para crear nuevos objetos se denomina constructor de objetos.

Funciones de flecha

Las funciones de flecha permiten una sintaxis corta para escribir expresiones de función.

No necesitas la functionpalabra clave, la returnpalabra clave y las **llaves** .

Ejemplo

```
// ES5
var x = function(x, y) {
 return x * y;
}

// ES6
const x = (x, y) => x * y;
```

Las funciones de flecha no tienen su propia función this. No son adecuadas para definir **métodos de objeto** .

Las funciones de flecha no se activan. Deben definirse **antes de** usarse.

Usar const es más seguro que usar var, porque una expresión de función siempre es un valor constante.

Solo se pueden omitir la returnpalabra clave y las llaves si la función es una sola sentencia. Por ello, conviene mantenerlas siempre:

Ejemplo

```
const x = (x, y) => { return x * y };
```

Las funciones de flecha no son compatibles con IE11 o versiones anteriores.