14 de febrero de 2025

Expresiones de función

En JavaScript, una función no es una "estructura mágica del lenguaje", sino un tipo de valor especial.

La sintaxis que usamos antes se llama *Declaración de Función*:

```
function sayHi() {
  alert( "Hola" );
}
```

Existe otra sintaxis para crear una función que se llama una *Expresión de Función*.

Esto nos permite crear una nueva función en el medio de cualquier expresión

Por ejemplo:

```
let sayHi = function() {
  alert( "Hola" );
};
```

Aquí podemos ver una variable `sayHi` obteniendo un valor —la nueva función— creada como `function() { alert("Hello"); }`.

Como la creación de una función ocurre en el contexto de una expresión de asignación, (el lado derecho de `=`), esto es una *Expresión de función*.

Note que no hay un nombre después de la palabra clave `function`. Omitir el nombre está permitido en las expresiones de función.

Aquí la asignamos directamente a la variable, así que el significado de estos ejemplos de código es el mismo: "crear una función y ponerla en la variable `sayHi`".

En situaciones más avanzadas, que cubriremos más adelante, una función puede ser creada e inmediatamente llamada o agendada para uso posterior, sin almacenarla en ningún lugar, permaneciendo así anónima.

La función es un valor

Reiteremos: no importa cómo es creada la función, una función es un valor. Ambos ejemplos arriba almacenan una función en la variable `sayHi`.

Incluso podemos mostrar aquel valor usando `alert`:

```
function sayHi() {
  alert( "Hola" );
}

alert( sayHi ); // muestra el código de la función
```

Tenga en cuenta que la última línea no ejecuta la función, porque no hay paréntesis después de `sayHi`. Existen lenguajes de programación en los que cualquier mención del nombre de una función causa su ejecución, pero JavaScript no funciona así.

En JavaScript, una función es un valor, por lo tanto podemos tratarlo como un valor. El código de arriba muestra su representación de cadena, que es el código fuente.

Por supuesto que es un valor especial, en el sentido que podemos invocarlo de esta forma `sayHi()`.

Pero sigue siendo un valor. Entonces podemos trabajar con ello como trabajamos con otro tipo de valores.

Podemos copiar una función a otra variable:

```
function sayHi() {   // (1) crear
  alert( "Hola" );
}

let func = sayHi;    // (2) copiar

func(); // Hola          // (3) ejecuta la copia (funciona)!
sayHi(); // Hola         // esto también funciona (por qué no lo haría)
```

Esto es lo que sucede arriba en detalle:

1. La Declaración de la Función `(1)` crea la función y la coloca dentro de la variable llamada `sayHi`.
2. Línea `(2)` copia la función en la variable `func`.
3. Ahora la función puede ser llamada de ambas maneras, `sayHi()` y `func()`.

También podríamos haber usado una expresión de función para declarar `sayHi` en la primera línea:

```
let sayHi = function() { // (1) crea
  alert( "Hola" );
};

let func = sayHi;  //(2)
// ...
```

Todo funcionaría igual.

¿Por qué hay un punto y coma al final?

Tal vez te preguntes por qué la Expresión de Función tiene un punto y coma `;` al final, pero la Declaración de Función no lo tiene:

```
function sayHi() {
  // ...
}

let sayHi = function() {
  // ...
};
```

La respuesta es simple: una expresión de función se crea aquí como `function(…) {…}` dentro de la sentencia de asignación `let sayHi = …;` . El punto y coma se recomienda para finalizar la sentencia, no es parte de la sintaxis de función.

El punto y coma estaría allí para una asignación más simple tal como `let sayHi = 5;` , y también está allí para la asignación de función.

Funciones Callback

Veamos más ejemplos del pasaje de funciones como valores y el uso de expresiones de función.

Escribimos una función `ask(question, yes, no)` con tres argumentos:

`question`

Texto de la pregunta

`yes`

Función a ejecutar si la respuesta es “Yes”

`no`

Función a ejecutar si la respuesta es “No”

La función deberá preguntar la `question` y, dependiendo de la respuesta del usuario, llamar `yes()` o `no()` :

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}

function showOk() {
  alert( "Estás de acuerdo." );
}

function showCancel() {
  alert( "Cancelaste la ejecución." );
}

// uso: las funciones showOk, showCancel son pasadas como argumentos de ask
ask("Estás de acuerdo?", showOk, showCancel);
```

En la práctica, tales funciones son bastante útiles. La mayor diferencia entre la función `ask` en la vida real y el ejemplo anterior es que las funciones de la vida real utilizan formas para interactuar con el usuario más complejas que un simple `confirme`. En el navegador, una función como tal normalmente dibuja una ventana de pregunta atractiva. Pero esa es otra historia.

Los argumentos de `ask` se llaman *funciones callback* o simplemente *callbacks*.

La idea es que pasamos una función y esperamos que se "devuelva la llamada" más tarde si es necesario. En nuestro caso, `showOk` se convierte en la callback para la respuesta "Yes", y `showCancel` para la respuesta "No".

Podemos usar Expresión de Función para redactar una función equivalente y más corta:

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}

ask(
  "Estás de acuerdo?",
  function() { alert("Estás de acuerdo"); },
  function() { alert("Cancelaste la ejecución."); }
);
```

Aquí, las funciones son declaradas justo dentro del llamado `ask(...)`. No tienen nombre, y por lo tanto se denominan *anónimas*. Tales funciones no se pueden acceder fuera de `ask` (porque no están asignadas a variables), pero eso es justo lo que queremos aquí.

Éste código aparece en nuestros scripts de manera muy natural, está en el archivo de comandos de JavaScript.

Una función es un valor representando una "acción"

Valores regulares tales como cadena de caracteres o números representan los *datos*.

Una función puede ser percibida como una *acción*.

La podemos pasar entre variables y ejecutarla cuando nosotros queramos.

Expresión de Función vs Declaración de Función

Formulemos las principales diferencias entre Declaración y Expresión de Funciones.

Primero, la sintaxis: cómo diferenciarlas en el código.

- *Declaración de Función:* una función, declarada como una instrucción separada, en el flujo de código principal.

```
1  // Declaración de Función
2  function sum(a, b) {
3    return a + b;
4  }
```

- *Expresión de Función:* una función, creada dentro de una expresión o dentro de otra construcción sintáctica. Aquí, la función es creada en el lado derecho de la "expresión de asignación" `=`:

```
1  // Expresión de Función
2  let sum = function(a, b) {
3    return a + b;
4  };
```

La diferencia más sutil es *cuándo* la función es creada por el motor de JavaScript.

Una Expresión de Función es creada cuando la ejecución la alcance y es utilizable desde ahí en adelante.

Una vez que el flujo de ejecución pase al lado derecho de la asignación `let sum = function…` – aquí vamos, la función es creada y puede ser usada (asignada, llamada, etc.) de ahora en adelante.

Las Declaraciones de Función son diferente.

Una Declaración de Función puede ser llamada antes de ser definida.

Por ejemplo, una Declaración de Función global es visible en todo el script, sin importar dónde se esté.

Esto se debe a los algoritmos internos. Cuando JavaScript se prepara para ejecutar el script, primero busca Declaraciones de Funciones globales en él y crea las funciones. Podemos pensar en esto como una "etapa de inicialización".

Y después de que se procesen todas las Declaraciones de Funciones, el código se ejecuta. Entonces tiene acceso a éstas funciones.

Por ejemplo, esto funciona:

```
sayHi("John"); // Hola, John

function sayHi(name) {
  alert( `Hola, ${name}` );
}
```

La Declaración de Función `sayHi` es creada cuando JavaScript está preparándose para iniciar el script y es visible en todas partes.

...Si fuera una Expresión de Función, entonces no funcionaría:

```
sayHi("John"); // error!

let sayHi = function(name) {  // (*) ya no hay magia
  alert( `Hola, ${name}` );
};
```

Las Expresiones de Función son creadas cuando la ejecución las alcance. Esto podría pasar solamente en la línea `(*)`. Demasiado tarde.

Otra característica especial de las Declaraciones de Funciones es su alcance de bloque.

En modo estricto, cuando una Declaración de Función se encuentra dentro de un bloque de código, es visible en todas partes dentro de ese bloque. Pero no fuera de él.

Por ejemplo, imaginemos que necesitamos declarar una función `welcome()` dependiendo de la variable `age` que obtengamos durante el tiempo de ejecución. Y luego planeamos usarlo algún tiempo después.

Si utilizamos la Declaración de Funciones, no funcionará como se esperaba:

```
let age = prompt("Cuál es tu edad?", 18);

// declarar condicionalmente una función
if (age < 18) {

  function welcome() {
    alert("Hola!");
  }

} else {

  function welcome() {
    alert("Saludos!");
  }

}

// ...usarla más tarde
welcome(); // Error: welcome no está definida
```

Esto se debe a que una Declaración de Función sólo es visible dentro del bloque de código en el que reside.

Aquí hay otro ejemplo:

```
let age = 16; // tomemos 16 como ejemplo

if (age < 18) {
  welcome();               // \   (corre)
                           //  |
  function welcome() {     //  |
    alert("¡Hola!");       //  |  La declaración de Función está disponible
  }                        //  |  en todas partes del bloque donde está decl
                           //  |
  welcome();               // /   (corre)

} else {

  function welcome() {
    alert("¡Saludos!");
  }
}

// Aquí estamos fuera de las llaves,
// por lo tanto no podemos ver la Declaración de Función realizada dentro de

welcome(); // Error: welcome no está definida
```

¿Qué podemos hacer para que `welcome` sea visible fuera de 'if'?

El enfoque correcto sería utilizar una Expresión de Función y asignar `welcome` a la variable que se declara fuera de 'if' y tiene la visibilidad adecuada.

Este código funciona según lo previsto:

```
let age = prompt("Cuál es tu edad?", 18);

let welcome;

if (age < 18) {

  welcome = function() {
    alert("Hola!");
  };

} else {

  welcome = function() {
    alert("Saludos!");
  };

}

welcome(); // ahora ok
```

O lo podemos simplificar aun más usando un operador de signo de pregunta `?` :

```
let age = prompt("¿Cuál es tu edad?", 18);

let welcome = (age < 18) ?
  function() { alert("¡Hola!"); } :
  function() { alert("¡Saludos!"); };

welcome(); // ahora ok
```

¿Cuándo debo elegir la Declaración de Función frente a la Expresión de Función?

Como regla general, cuando necesitamos declarar una función, la primera que debemos considerar es la sintaxis de la Declaración de Función. Da más libertad en cómo organizar nuestro código, porque podemos llamar a tales funciones antes de que sean declaradas.

También es un poco más fácil de buscar `function f(…) {…}` en el código comparado con `let f = function(…) {…}` . La Declaración de Función es más llamativa.

...Pero si una Declaración de Función no nos conviene por alguna razón, o necesitamos declaración condicional (hemos visto un ejemplo), entonces se debe usar la Expresión de función.

Resumen

- Las funciones son valores. Se pueden asignar, copiar o declarar en cualquier lugar del código.
- Si la función se declara como una declaración separada en el flujo del código principal, eso se llama "Declaración de función".
- Si la función se crea como parte de una expresión, se llama "Expresión de función".
- Las Declaraciones de Funciones se procesan antes de ejecutar el bloque de código. Son visibles en todas partes del bloque.
- Las Expresiones de Función se crean cuando el flujo de ejecución las alcanza.

En la mayoría de los casos, cuando necesitamos declarar una función, es preferible una Declaración de Función, ya que es visible antes de la declaración misma. Eso nos da más flexibilidad en la organización del código, y generalmente es más legible.

Por lo tanto, deberíamos usar una Expresión de Función solo cuando una Declaración de Función no sea adecuada para la tarea. Hemos visto un par de ejemplos de eso en este capítulo, y veremos más en el futuro.

Lección anterior | Próxima lección

Compartir

Mapa del Tutorial

Comentarios

- Si tiene sugerencias sobre qué mejorar, por favor enviar una propuesta de GitHub o una solicitud de extracción en lugar de comentar.
- Si no puede entender algo en el artículo, por favor explique.
- Para insertar algunas palabras de código, use la etiqueta `<code>`, para varias líneas – envolverlas en la etiqueta `<pre>`, para más de 10 líneas – utilice una entorno controlado (sandbox) (plnkr, jsbin, codepen...)

9 Comentarios

Acceder

G

Únete a la conversación...

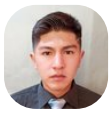
INICIAR SESIÓN CON

O REGISTRARSE CON DISQUS ?

Nombre

19 Comparte

Mejores Más recientes Más antiguos



Juan Daniel MQ

hace 3 años

Es grato ver código simplificado y limpio mas que todo en la Expresión de funcion.

3 1 Responder

Paola Torres

hace 3 años

Esta muy bien explicado este tema, por fin se la diferencia entre una expresión de función y una declaración de función.

1 0 Responder

J

Jose

hace 3 años

Uffff, esto cada vez se complica más, pero estoy encantado con este maravilloso tutorial. Mil gracias.

1 0 Responder

Juan Camilo Rodriguez Portilla

hace 10 meses

Ni sabía que se llamaban así ese tipo de funciones, muchas gracias por la explicación que a mi opinion hace entender al más junior

0 0 Responder

Gustavo Redondo

hace 2 años

Excelente explicación, la simplicidad del código es el éxito de la programación.

0 0 Responder

J

Juan Daniel Zea Arango

hace 2 años

Gracias!!
Me encanto el tutorial

0 0 Responder

Felipe Arce
hace 3 años

Muchas gracias por la explicacion

0 0 Responder

A

Artugrol Ousmane
hace 4 años

merci, شكرا

0 1 Responder

Christian Chang
hace 4 años

Muchas gracias por la explicación 🙄

0 1 Responder