3 de julio de 2022

Funciones Flecha, lo básico

Hay otra sintaxis muy simple y concisa para crear funciones, que a menudo es mejor que las Expresiones de funciones.

Se llama "funciones de flecha", porque se ve así:

```
let func = (arg1, arg2, ..., argN) => expression;
```

Esto crea una función `func` que acepta los parámetros `arg1..argN`, luego evalúa la `expression` del lado derecho mediante su uso y devuelve su resultado.

En otras palabras, es la versión más corta de:

```
let func = function(arg1, arg2, ..., argN) {
  return expression;
};
```

Veamos un ejemplo concreto:

```
let sum = (a, b) => a + b;

/* Esta función de flecha es una forma más corta de:

let sum = function(a, b) {
  return a + b;
};
*/

alert( sum(1, 2) ); // 3
```

Como puedes ver, `(a, b) => a + b` significa una función que acepta dos argumentos llamados `a` y `b`. Tras la ejecución, evalúa la expresión `a + b` y devuelve el resultado.

- Si solo tenemos un argumento, se pueden omitir paréntesis alrededor de los parámetros, lo que lo hace aún más corto.

  Por ejemplo:

```
let double = n => n * 2;
// Más o menos lo mismo que: let double = function(n) { return n * 2 }

alert( double(3) ); // 6
```

- Si no hay parámetros, los paréntesis estarán vacíos; pero deben estar presentes:

```
let sayHi = () => alert("¡Hola!");

sayHi();
```

Las funciones de flecha se pueden usar de la misma manera que las expresiones de función.

Por ejemplo, para crear dinámicamente una función:

```
let age = prompt("What is your age?", 18);

let welcome = (age < 18) ?
  () => alert('¡Hola!') :
  () => alert("¡Saludos!");

welcome();
```

Las funciones de flecha pueden parecer desconocidas y poco legibles al principio, pero eso cambia rápidamente a medida que los ojos se acostumbran a la estructura.

Son muy convenientes para acciones simples de una línea, cuando somos demasiado flojos para escribir muchas palabras.

Funciones de flecha multilínea

Las funciones de flecha que estuvimos viendo eran muy simples. Toman los parámetros a la izquierda de `=>`, los evalúan y devuelven la expresión del lado derecho.

A veces necesitamos una función más compleja, con múltiples expresiones o sentencias. En ese caso debemos encerrarlos entre llaves. La diferencia principal es que las llaves necesitan usar un `return` para devolver un valor (tal como lo hacen las funciones comunes).

Como esto:

```
let sum = (a, b) => {  // la llave abre una función multilínea
  let result = a + b;
  return result; // si usamos llaves, entonces necesitamos un "return" explíc
};

alert( sum(1, 2) ); // 3
```

Más por venir

Aquí elogiamos las funciones de flecha por su brevedad. ¡Pero eso no es todo!

Las funciones de flecha tienen otras características interesantes.

Para estudiarlas en profundidad, primero debemos conocer algunos otros aspectos de JavaScript, por lo que volveremos a las funciones de flecha más adelante en el capítulo Funciones de flecha revisadas.

Por ahora, ya podemos usar las funciones de flecha para acciones de una línea y devoluciones de llamada.

Resumen

Las funciones de flecha son útiles para acciones simples, especialmente las de una sola línea. Vienen en dos variantes:

1. Sin llaves: `(...args) => expression` – el lado derecho es una expresión: la función la evalúa y devuelve el resultado. Pueden omitirse los paréntesis si solo hay un argumento, por ejemplo `n => n*2`.
2. Con llaves: `(...args) => { body }` – las llaves nos permiten escribir varias declaraciones dentro de la función, pero necesitamos un `return` explícito para devolver algo.

Tareas

Reescribe con funciones de flecha

Reemplace las expresiones de función con funciones de flecha en el código a continuación:

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

ask(
  "Do you agree?",
  function() { alert("You agreed."); },
  function() { alert("You canceled the execution."); }
);
```

solución

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

ask(
  "Do you agree?",
  () => alert("You agreed."),
  () => alert("You canceled the execution.")
);
```

Se ve corto y limpio, ¿verdad?

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...)

Julio César

hace 3 años

let ask = (question, yes, no) => {
(confirm(question)) ? yes() : no()
}

ask(
"Do you agree?",
() => alert("You agreed."),
() => alert("You canceled the execution.")
);

2 0 Responder

H

Hector Horacio Anzola Rubio

hace 3 años

o hice asi

let ask = (question, yes, no) => confirm(question)? yes(): no();

ask(

"Do you agree?",

function() { alert("You agreed."); },

function() { alert("You canceled the execution."); }

);

1 0 Responder

David SH → Hector Horacio Anzola Rubio

hace 3 años

x2

0 0 Responder

Sebastien

hace 4 años editado

i hope it helps you

Vista – uploads.disquscdn.com

1 0 Responder

O

Oscar Florez

hace 16 días

```
let ask = (question, yes, no) => {
if (confirm(question)) yes();
else no();
};

ask(
"Do you agree?",
() => alert("You agreed."),
() => alert("You canceled the execution.")
);
```

0 0 Responder

C

Cristian

hace 2 meses

Solucion mia:

```
let ask = (question, yes, no) => {
if (confirm(question)) yes();
else no();
};

ask("Do you agree?",
() => alert("You agreed"),
() => alert("You canceled the execution.")
);
```

0 0 Responder

wolf code

hace 2 años

Buenaso este curso, yo le resolví de esta manera

Vista – uploads.disquscdn.com

0 0 Responder

J

Jhan Carlos

hace 3 años

Podriamos decir que una "funcion flecha" es otra forma de escribir una "expresion de funcion" ?

0 0 Responder

C **Christopher Gallardo** → Jhan Carlos

hace 4 meses

Se usa solo para expresiones , a menudo se usa para mejorar expresiones de funciones, no se usa para declarar una función

1 0 Responder

H **Hector Horacio Anzola Rubio** → Jhan Carlos

hace 3 años

Me hago la misma pregunta. La funcion flecha es una expresion de funcion o una declaracion de funcion

0 0 Responder

M **Mynor Morales**

hace 3 años

```
let ask = (question, yes, no) => (confirm(question)) ? yes() : no();
ask("Do you agree?", () => alert("You agreed"), () => alert("You canceled"))
```

0 0 Responder

 **dharma thorki**

hace 3 años

let ask=(question, x, y)=>confirm(question)? alert('you agreeed'):alert('you canceled execution');

ask('do you agree')

0 0 Responder

 **dharma thorki** → dharma thorki

hace 3 años

let ask=(question)=>confirm(question)? alert('you agreeed'):alert('you canceled execution');

ask('do you agree')

0 0 Responder

M **Mynor Morales**

hace 3 años

let ask = (question, yes, no) => (confirm(question))?yes():no()
ask ("Do you agree?", ()=>alert("You agreed."), ()=>alert("You canceled the execution."))

0 0 Responder

 **Carlos Alberto Rimachi Silva**

hace 4 años

Yo lo hice de esta manera. Vista – uploads.disquscdn.com

0 0 Responder

P **paco de lucia**
hace 4 años

Facilito

0 0 Responder

 **Oliver Atom Castro**
hace 4 años

let ask = (question, yes, no) => (confirm(question)) ? yes() : no ();

ask (" Do you agree? ", () => alert ("You agreed. "), () => alert (" You canceled the execution."));

0 0 Responder

A **Alberto**
hace 4 años

Vista – uploads.disquscdn.com

0 0 Responder

 **Yelitza Suniaga**
hace 4 años

//Con operador ternario en 2 líneas
const ask = (question, yes, no) => (confirm(question))? yes(): no();

ask("Do you agree",()=>alert("you agreed"),()=>alert("you canceled the execution"))

0 0 Responder

A **Artugrol Ousmane** Yelitza Suniaga
hace 4 años

best

0 0 Responder

P **paco de lucia** Yelitza Suniaga
hace 4 años

venga chula

0 0 Responder

 **giorgi**
hace 5 años

Vista – uploads.disquscdn.com

Tengo el siguiente código. En la línea 15 utilizo la función isPrime como argumento para la función filter. ¿Por qué funciona? Es decir solo envio la función isPrime. Qué hace que funcione?

0 0 Responder

TutosCarlos → giorgi

hace 4 años editado

Normalmente los metodos como el filter de los arrays reciben un arrow function o una expresión de una función, ejemplo:

```
// Callback como expresión de una función
const result = array.filter(function(item ) {
return item > 5;
});

// Callback como un arrow function (anonima)
const result = array.filter(item => item > 5);
```

O en tu caso que recibe una función.
En el código que muestras pasas la función isPrime a el método filter, que sería lo mismo que tener esto:

```
const primeNumber = array.filter(number => isPrime(number));
```

Siendo algo redundante tener **number** como parámetro y pasarlo a la función isPrime, se puede **abreviar** pasando solo la función sin parantesis que de manera implícita recibe el parámetro **number**.

```
const primeNumber = array.filter(isPrime);
```

Siendo el código aún más legible.

0 0 Responder

El comentario ha sido eliminado.

Lucas Frezzini → Guest

hace 5 años editado

Incluso podes aplicar el operador ternario y que quede más sencillo visualmente:

```
let ask = (question, yes, no) => {
    (confirm(question)) ? yes() : no();
}
```

O sin las llaves en una sola linea también, aunque se pierde quizás un poco la lectura:

```
let ask = (question, yes, no) => (confirm(question)) ? yes() : no();
```

6 0 Responder

Mauro Maidana → Guest

contáctenos