Módulo Javascript

Funciones Callback de Arrays

Objetivos: Conocer las funciones de utilidad de Arrays que utilizan callbacks.

Funciones de arrays con callbacks

Las siguientes funciones son herramientas útiles en el kit de herramientas de cualquier desarrollador de JavaScript. La función .find() se usa para buscar elementos en un array, .map() para transformar elementos en un array, .filter() para seleccionar elementos en un array, y .reduce() para reducir un array a un único valor.

La función .find()

La función .find() devuelve el primer elemento en un array que cumple con cierta condición. Si no encuentra ningún elemento que cumpla con la condición, entonces devuelve 'undefined'.

Ejemplo:

```JavaScript
const numbers = [1, 2, 3, 4, 5];

const evenNumber = numbers.find((number) => number % 2 === 0);

console.log(evenNumber); // 2
```

En este ejemplo, la función .find() busca el primer número par en el array numbers. Al encontrar el número 2, la función lo devuelve.

La función .map()

La función .map() crea un nuevo array con los resultados de la aplicación de una función dada a cada elemento del array original.

Ejemplo:

```javascript
const numbers = [1, 2, 3, 4, 5];

const squaredNumbers = numbers.map((number) => number ** 2);

console.log(squaredNumbers); // [1, 4, 9, 16, 25]
```

En este ejemplo, la función .map() eleva al cuadrado cada elemento del array numbers y devuelve el nuevo array squaredNumbers.

La función .filter()

La función .filter() crea un nuevo array con todos los elementos que cumplan con cierta condición. Si no se encuentra ningún elemento que cumpla con la condición, entonces devuelve un array vacío.

Ejemplo:

```javascript
const numbers = [1, 2, 3, 4, 5];

const evenNumbers = numbers.filter((number) => number % 2 === 0);

console.log(evenNumbers); // [2, 4]
```

En este ejemplo, la función .filter() crea un nuevo array con todos los números pares del array original numbers.

La función .reduce()

La función .reduce() ejecuta una función reductora en cada elemento del array para devolver un único valor. La función reductora toma dos argumentos: el acumulador y el valor actual.

Ejemplo:

```javascript
const numbers = [1, 2, 3, 4, 5];

const sum = numbers.reduce((accumulator, currentValue) =>
accumulator + currentValue);

console.log(sum); // 15
```

En este ejemplo, la función .reduce() devuelve la suma de todos los números en el array numbers.

La función .every()

La función .every() verifica si todos los elementos en un array cumplen con una determinada condición. Devuelve true si todos los elementos pasan la prueba, y false si al menos uno de los elementos no la cumple.

Ejemplo:

```javascript
const numbers = [2, 4, 6, 8, 10];

const allAreEven = numbers.every((number) => number % 2 === 0);

console.log(allAreEven); // true
```

En este ejemplo, la función .every() verifica si todos los elementos del array numbers son pares. Como todos son pares, la función devuelve true.

La función .some()

La función .some() verifica si al menos uno de los elementos en un array cumple con una determinada condición. Devuelve true si al menos un elemento cumple la condición, y false si ninguno lo hace.

Ejemplo:

```javascript
const numbers = [1, 3, 5, 7, 8];

const hasEvenNumber = numbers.some((number) => number % 2 === 0);

console.log(hasEvenNumber); // true
```

En este ejemplo, la función .some() verifica si al menos un elemento del array numbers es par. Como el número 8 es par, la función devuelve true.

Combinando .every() y .some()

Estas dos funciones pueden combinarse para verificar si todos los elementos en un array cumplen con una condición excepto uno o más elementos que no la cumplen. Esto se puede hacer utilizando .every() en combinación con .some().

Ejemplo:

```javascript
const numbers = [2, 4, 6, 7, 8];

const allAreEvenExceptOne = numbers.every((number) => number % 2
=== 0) && numbers.some((number) => number % 2 !== 0);

console.log(allAreEvenExceptOne); // true
```

En este ejemplo, la función .every() verifica si todos los elementos del array numbers son pares, excepto uno que es impar. Luego, la función .some() verifica si al menos hay un elemento impar en el array. Como ambas condiciones se cumplen, la función devuelve true.

En conclusión, estas dos funciones son muy útiles para verificar condiciones en un array y devolver un valor booleano indicando si se cumple o no la condición. La función .every() devuelve true si todos los elementos en el array pasan la prueba, mientras que .some() devuelve true si al menos uno de los elementos cumple la condición.