Javascript

Carrera Programador full-stack

Callbacks

Qué veremos!!

Tema principal funciones callbacks en Javascript

- Funciones sincrónicas y sus límites
- Asincronismo
- Que es un callback?
- Porque utilizarlas?.
- Gestión de errores
- Problemas comunes

Funciones sincrónicas

Hasta ahora las funciones que utilizamos se ejecutan de forma síncrona esto significa que se sigue una secuencia de ejecución y las instrucciones se ejecutan una después de otra.

Funciones sincrónicas

```
function suma(a, b) {
  return a + b;
}
console.log(suma(3, 5));
// ## Resultado en pantalla ##
// 8
```

En el ejemplo vemos una función normal, donde el resultado de la operación es devuelto usando la instrucción return; esto es llamado el estilo directo, y es la forma más común de devolver un resultado en una operación síncrona.

Funciones callback

El equivalente al ejemplo anterior usando una función callback.

```
function suma(a, b) {
 return a + b;
}

function oper(a, b, callback) {
 console.log(callback(a, b));
}

console.log('Antes de la ejecucion');
oper(3, 5, suma);
console.log('Despues de la ejecucion');

// ## Resultado en pantalla ##
// Antes de la ejecucion
// 8
// Despues de la ejecucion
```

Con este ejemplo se recibe suma con el nombre callback y se ejecuta

Se envía la función suma

Funciones callback

```
function resta(a, b) {
  return a - b;
}

function oper(a, b, callback) {
  console.log(callback(a, b));
}

console.log('Antes de la ejecucion');

oper(8, 5, resta);

console.log('Despues de la ejecucion');

// ## Resultado en pantalla ##
// Antes de la ejecucion
// 3
// Despues de la ejecucion
```

Con este ejemplo se recibe resta con el nombre callback y se ejecuta

Se envía la función resta

Funciones callback

```
function oper(a, b, operacion) {
  console.log(operacion(a, b));
}
console.log('Antes de la ejecucion');

oper(3, 5, function (a, b) {
    return a + b;
});

console.log('Despues de la ejecucion');
// ## Resultado en pantalla ##
// Antes de la ejecucion
// 8
// Despues de la ejecucion
```

No es necesario que el parámetro de la función se llame callback (puede tener cualquier nombre)

Cambiamos utilizando función anónima

Funciones callback

```
function oper(a, b, callback) {
  console.log(callback(a, b));
}
console.log('Antes de la ejecucion');

oper(3, 5, (a, b) => a + b);

console.log('Despues de la ejecucion');
// ## Resultado en pantalla ##
// Antes de la ejecucion
// 8
// Despues de la ejecucion
```

Con funciones flecha

Funciones sincrónicas

Esta forma de ver los códigos en la vida real sería como:

Esperando que se lave la ropa

Esperando que se laven los platos

Esperando que se cocine la comida

Funciones asincrónicas

Pero la realidad es que los lenguajes permiten la ejecución de sentencias de manera asincrónica esto significa que se pueden ejecutar instrucciones en paralelo, es decir “al mismo tiempo”.

Para realizar esta acción Javascript lo hace permitiendo enviar una función como parámetro de otra (llamados callbacks).

Funciones asincrónicas

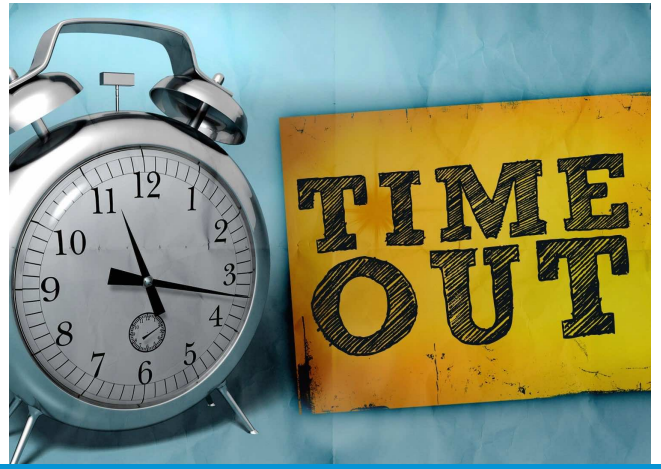
```
function oper(a, b, callback) {
  setTimeout(function() {
    console.log(callback(a, b));
  }, 500);
}
console.log('Antes de la ejecucion');

oper(3, 10, (a, b) => a + b);

console.log('Despues de la ejecucion?');
// ## Resultado en pantalla ##
// Antes de la ejecucion
// Despues de la ejecucion?
// 13
```

Lo mismo que utilizamos antes pero agregando un retraso en la ejecución del callback

Asincronismo con setTimeout

En el ejemplo anterior cambia el orden de la ejecución del código, mostrando al final el resultado de la suma. Usamos la función `setTimeout` para simular una operación asíncrona.

La función `setTimeout` toma como primer parámetro una función callback y como segundo parámetro el tiempo de demora en ejecutar el contenido de la función callback.



Asincronismo con setTimeout 2

Como `setTimeout` es una función asíncrona, no espera para ejecutar el código de la función callback; lo que hace es devolver el control a la función `oper`, y esta sigue la ejecución del programa, dejando la función callback a la espera en el ciclo de eventos de javascript para ser ejecutada.

Este es el funcionamiento por defecto de como javascript maneja las operaciones asíncronas.

Si quieres saber más sobre el ciclo de eventos de javascript puedes visitar:
https://developer.mozilla.org/es/docs/Web/JavaScript/EventLoop