



UNIVERSIDAD GERARDO BARRIOS

SEDE CENTRAL SAN MIGUEL /CENTRO REGIONAL USulután

Datos Generales	
Facultad	Ciencia y Tecnología
Asignatura	Programación Computacional IV
Docente	
No. de Unidad	3
Contenido a desarrollar	Asincronismo en Node.js

DESARROLLO DE CONTENIDOS

Asincronismo en Node.js

La programación asíncrona es una de las grandes ventajas de Javascript y uno de los grandes secretos de la gestión de la concurrencia de Node.js, pero también es fuente de problemas, errores y confusiones, tanto para los programadores que están empezando, como para los más expertos.

Uno de los primeros conceptos que debemos incorporar cuando empezamos a programar en Node.js es la idea de asincronismo. En Node.js todas las operaciones de entrada/salida (IO) se ejecutan de manera asíncrona y no bloqueante

Esto tiene muchas ventajas en cuanto a la optimización de los recursos (permite manejar muchas operaciones IO en un mismo thread de forma eficiente), pero al mismo tiempo nos obliga a adoptar algunos patrones de programación diferentes a otras plataformas mas tradicionales que usan IO bloqueante (como Java o PHP).

Función asíncrona de ejemplo

Primero vamos a definir una sencilla función asíncrona que nos servirá para mostrar las diferentes aproximaciones sobre el control. No es una función útil, simplemente calcula el cuadrado de un número de forma asíncrona y además retrasa de forma aleatoria su ejecución por medio de `setTimeout()` y `Math.random()`.

```
"use strict";
function asyncSqrt(value, callback) {
  console.log('START execution with value =', value);
  setTimeout(function() {
    callback(value, value * value);
  }, 0 | Math.random() * 100);
}

asyncSqrt(2, function(value, result) {
  console.log('END execution with value =', value, 'and result =', result);
});
console.log('COMPLETED ?');
```

puedes ver el código en ejecución en <https://replit.com/@todojs/1-asyncSqrt>

```
START execution with value = 0
START execution with value = 1
START execution with value = 2
START execution with value = 3
START execution with value = 4
START execution with value = 5
START execution with value = 6
START execution with value = 7
START execution with value = 8
START execution with value = 9
COMPLETED ?
Hint: hit control+c anytime to enter REPL.
END execution with value = 8 and result = 64
END execution with value = 5 and result = 25
END execution with value = 0 and result = 0
END execution with value = 9 and result = 81
END execution with value = 6 and result = 36
END execution with value = 2 and result = 4
END execution with value = 1 and result = 1
END execution with value = 4 and result = 16
END execution with value = 7 and result = 49
```

El resultado de ejecutar este código puede desconcertar un poco, ya que si fuera un código síncrono esperaríamos ver los mensajes STAR y END uno detrás de otro y

terminados por COMPLETED, pero al ser un código asíncrono el resultado es completamente diferente.

Lo que ha pasado es que se ha ejecutado el bucle for con cada una de las llamadas a `asynSqrt()` y directamente se ha pasado a ejecutar el mensaje COMPLETED, ya que todas las llamadas a nuestra función se ejecutan de forma asíncrona. Los resultados de esta ejecución no siguen ningún orden, ya que cada una de estas llamadas concluye sin respetar el orden de llamada.

La asincronía es uno de los pilares fundamentales de Javascript. El objetivo de esta guía es profundizar en las piezas y elementos que la hacen posible. Teniendo claro estos conceptos, podrás ponerlos en práctica en tu código y escribir mejores aplicaciones.

Las funciones anónimas son aquellas que se crean sin nombre

Vamos a escribir y probar el siguiente código. Primero debes descargar e instalar Node.js en la siguiente dirección: <https://nodejs.org/es/>

Para probar el asincronismo, utilizaremos una web que nos proporciona una api rest json <https://ghibliapi.herokuapp.com/films>

JSON	Datos sin procesar	Cabeceras
Guardar	Copiar	Contraer todo Expandir todo
		🔍 title
▼ 0:		
title:		"Castle in the Sky"
original_title:		"天空の城ラピュタ"
original_title_romanised:		"Tenkū no shiro Rapyuta"
▼ 1:		
title:		"Grave of the Fireflies"
original_title:		"火垂るの墓"
original_title_romanised:		"Hotaru no haka"
▼ 2:		
title:		"My Neighbor Totoro"
original_title:		"となりのトトロ"
original_title_romanised:		"Tonari no Totoro"
▼ 3:		
title:		"Kiki's Delivery Service"
original_title:		"魔女の宅急便"
original_title_romanised:		"Majo no takkyūbin"
▼ 4:		
title:		"Only Yesterday"
original_title:		"おもひでぽろぽろ"
original_title_romanised:		"Omoide poro poro"
▼ 5:		
title:		"Porco Rosso"
original_title:		"紅の豚"
original_title_romanised:		"Kurenai no buta"
▼ 6:		

Primero instalaremos nodejs



Desde la consola de msdos creamos una carpeta en el lugar donde prefieras, digamos que seleccionas mis documentos, entramos a esa carpeta y tecleamos en la consola: **npm init -y**

A continuación, instale la biblioteca request: **npm i request --save**

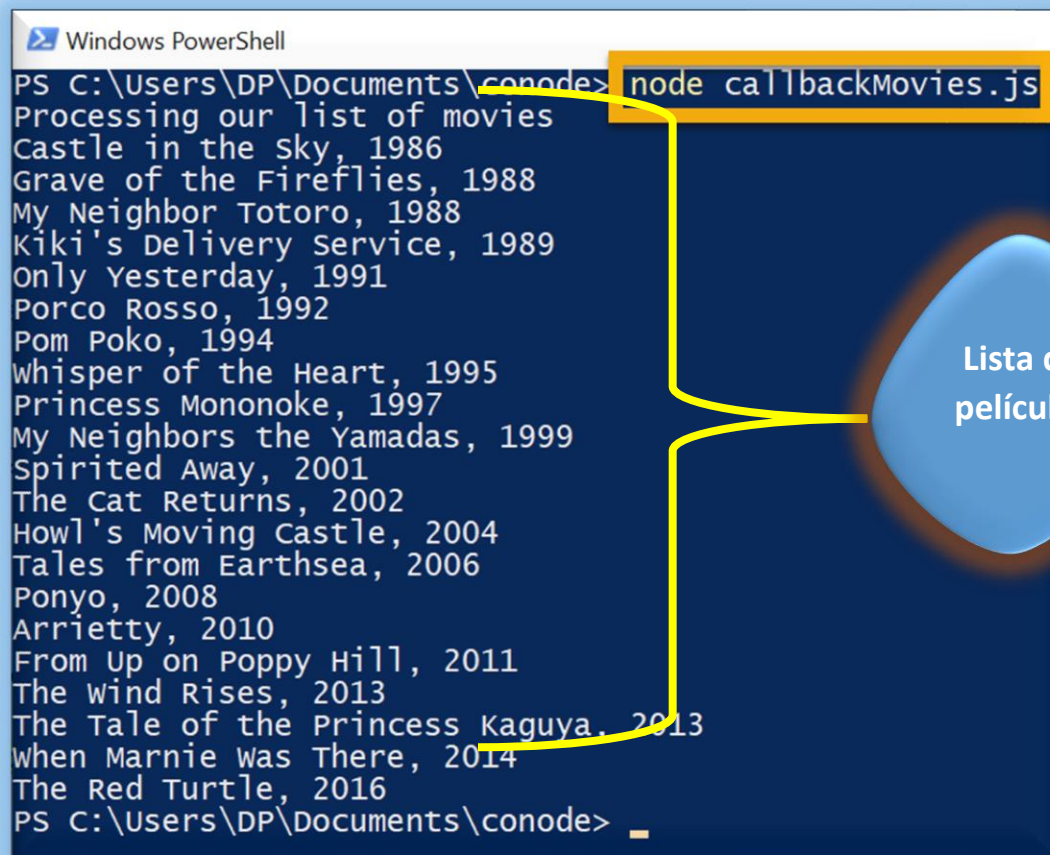
Ahora, dentro de la carpeta, vamos a crear un archivo llamado **callbackMovies.js** y lo abrimos con un editor, por ejemplo sublimetext, vamos a escribir:

```
const request = require('request');
request('https://ghibliapi.herokuapp.com/films', (error, response, body) => {
  if (error) {
    console.error(`Could not send request to API: ${error.message}`);
    return;
  }
  if (response.statusCode !== 200) {
    console.error(`Expected status code 200 but received ${response.statusCode}.`);
    return;
  }
  console.log('Processing our list of movies');
  movies = JSON.parse(body);
  movies.forEach(movie => {
    console.log(`${movie['title']}, ${movie['release_date']}`);
  });
});
```

```
});  
});
```

Guardamos y vamos a la consola y tecleamos:

node callbackMovies.js



```
Windows PowerShell  
PS C:\Users\DP\Documents\conode> node callbackMovies.js  
Processing our list of movies  
Castle in the sky, 1986  
Grave of the Fireflies, 1988  
My Neighbor Totoro, 1988  
Kiki's Delivery Service, 1989  
Only Yesterday, 1991  
Porco Rosso, 1992  
Pom Poko, 1994  
whisper of the Heart, 1995  
Princess Mononoke, 1997  
My Neighbors the Yamadas, 1999  
Spirited Away, 2001  
The Cat Returns, 2002  
Howl's Moving Castle, 2004  
Tales from Earthsea, 2006  
Ponyo, 2008  
Arrietty, 2010  
From Up on Poppy Hill, 2011  
The wind Rises, 2013  
The Tale of the Princess Kaguya, 2013  
when Marnie was There, 2014  
The Red Turtle, 2016  
PS C:\Users\DP\Documents\conode> _
```

Lista de
películas

En nuestra función de devolución de llamada existen tres argumentos: error, response y body. Cuando la solicitud HTTP se completa, se otorgan automáticamente valores a los argumentos dependiendo del resultado. Si la solicitud no se enviara, error contendría un objeto. Sin embargo, response y body tendrían el valor null. Si la solicitud se realizó correctamente, la respuesta HTTP se almacenará en response. Si en nuestra respuesta HTTP se muestran datos (en este ejemplo obtenemos JSON), los datos se establecen en body.

Obtuvimos con éxito una lista de películas de Studio Ghibli con el año en que se estrenaron. Ahora, terminaremos este programa escribiendo la lista de películas que actualmente registramos en un archivo. En lugar de mostrar la lista de películas en la consola, la guardaremos en un archivo csv de Excel.

Actualiza el código del archivo **callbackMovies.js** a:

```
const request = require('request');
const fs = require('fs');

request('https://ghibliapi.herokuapp.com/films', (error, response, body) => {
  if (error) {
    console.error(`Could not send request to API: ${error.message}`);
    return;
  }

  if (response.statusCode !== 200) {
    console.error(`Expected status code 200 but received ${response.statusCode}.`);
    return;
  }

  console.log('Processing our list of movies');
  movies = JSON.parse(body);
  let movieList = '';
  movies.forEach(movie => {
    movieList += `${movie['title']}, ${movie['release_date']}\n`;
  });

  fs.writeFile('callbackMovies.csv', movieList, (error) => {
    if (error) {
      console.error(`Could not save the Ghibli movies to a file: ${error}`);
      return;
    }

    console.log('Lista de películas guardada en callbackMovies.csv');
  });
});
```

Guardamos y lo volvemos a correr:

Windows PowerShell

```
PS C:\Users\DP\Documents\conode> node callbackMovies.js
```

Al presionar enter veremos:

Windows PowerShell

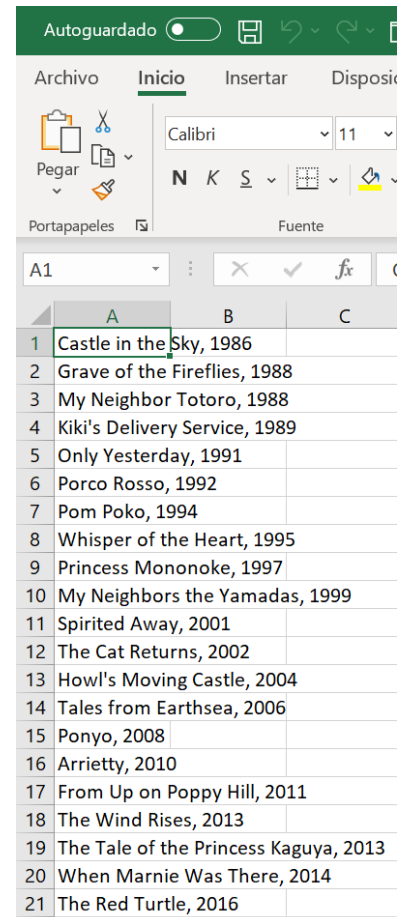
```
PS C:\Users\DP\Documents\conode> node callbackMovies.js
Processing our list of movies
Lista de películas guardada en callbackMovies.csv
PS C:\Users\DP\Documents\conode>
```

Si abrimos la carpeta que creamos en mis documentos:

Nombre

- ~
- node_modules
- callbackMovies.csv**
- callbackMovies.js
- package.json
- package-lock.json

Puedes notar que se creó un archivo callbackMovies.csv, si abrimos ese archivo veremos:



The screenshot shows a Microsoft Word document with a table containing a list of movies. The table has three columns: A, B, and C. The first column (A) contains a list of movie titles and years, numbered 1 through 21. The second column (B) is empty. The third column (C) is empty. The table is titled 'Autoguardado' (Autosaved) in the top left corner. The font is Calibri, size 11.

A	B	C
1 Castle in the Sky, 1986		
2 Grave of the Fireflies, 1988		
3 My Neighbor Totoro, 1988		
4 Kiki's Delivery Service, 1989		
5 Only Yesterday, 1991		
6 Porco Rosso, 1992		
7 Pom Poko, 1994		
8 Whisper of the Heart, 1995		
9 Princess Mononoke, 1997		
10 My Neighbors the Yamadas, 1999		
11 Spirited Away, 2001		
12 The Cat Returns, 2002		
13 Howl's Moving Castle, 2004		
14 Tales from Earthsea, 2006		
15 Ponyo, 2008		
16 Arrietty, 2010		
17 From Up on Poppy Hill, 2011		
18 The Wind Rises, 2013		
19 The Tale of the Princess Kaguya, 2013		
20 When Marnie Was There, 2014		
21 The Red Turtle, 2016		

Observando los cambios resaltados, podemos ver que importamos el módulo fs. Este módulo forma parte de la configuración estándar de todas las instalaciones de Node.js y contiene un método writeFile() que puede escribir en un archivo de forma asíncrona.

Es importante observar que escribimos en nuestro archivo CSV en la devolución de llamada de la solicitud HTTP. Una vez que el código se encuentra en la función de devolución de llamada, solo escribirá en el archivo después de que se haya completado la solicitud HTTP. Si quisiéramos comunicarnos con una base de datos después de escribir nuestro archivo CSV, crearíamos otra función asíncrona que se invocaría en la devolución de llamada de nuestro writeFile(). Mientras más código asíncrono tengamos, más funciones de devolución de llamada deberán anidarse.

Cuando en las devoluciones de llamada anidadas hay muchas líneas de código para ejecutar, se vuelven considerablemente más complejas e ilegibles. A medida que aumenten el tamaño y la complejidad de su proyecto de JavaScript, este efecto se hará más pronunciado hasta que finalmente no pueda manejarse. Debido a esto, los desarrolladores ya no utilizan las devoluciones de llamada para manejar las operaciones asíncronas. Para mejorar la sintaxis de nuestro código asíncrono, podemos usar promesas como alternativa.

Usar promesas para la programación asíncrona concisa

Una promesa es un objeto de JavaScript en el que se mostrará un valor en algún momento del futuro. En las funciones asíncronas se pueden mostrar objetos de promesas en lugar de valores concretos. Si obtenemos un valor en el futuro, afirmamos que la promesa se cumplió. Si obtenemos un error en el futuro, afirmamos que la promesa se rechazó. De lo contrario, se siguen realizando tareas vinculadas a la promesa en un estado de operación pendiente.

Las promesas suelen adoptar la siguiente forma:

```
promiseFunction()  
  .then([ Callback Function for Fulfilled Promise ])  
  .catch([ Callback Function for Rejected Promise ])
```

Axios es un cliente HTTP basado en promesas para JavaScript; lo instalaremos:

```
npm i axios --save
```


cree un archivo nuevo promiseMovies.js


y escribimos:

```
const axios = require('axios');
const fs = require('fs').promises;

axios.get('https://ghibliapi.herokuapp.com/films')
  .then((response) => {
    console.log('Successfully retrieved our list of movies');
    response.data.forEach(movie => {
      console.log(`${movie['title']}, ${movie['release_date']}`);
    });
  })
```

En powershell escribimos **node promiseMovies.js**

Y veremos:

 Windows PowerShell

```
PS C:\Users\DP\Documents\conode> node promiseMovies.js
Successfully retrieved our list of movies
Castle in the sky, 1986
Grave of the Fireflies, 1988
My Neighbor Totoro, 1988
Kiki's Delivery Service, 1989
Only Yesterday, 1991
Porco Rosso, 1992
Pom Poko, 1994
Whisper of the Heart, 1995
Princess Mononoke, 1997
My Neighbors the Yamadas, 1999
Spirited Away, 2001
The Cat Returns, 2002
Howl's Moving Castle, 2004
Tales from Earthsea, 2006
Ponyo, 2008
Arrietty, 2010
From Up on Poppy Hill, 2011
The Wind Rises, 2013
The Tale of the Princess Kaguya, 2013
When Marnie Was There, 2014
The Red Turtle, 2016
PS C:\Users\DP\Documents\conode>
```

Explicación: Después de realizar una solicitud HTTP GET con `axios.get()`, usamos la función `then()`, que se ejecuta solo cuando la promesa se cumple. En este caso, imprimimos las películas en la pantalla al igual que en el ejemplo de devoluciones de llamadas.

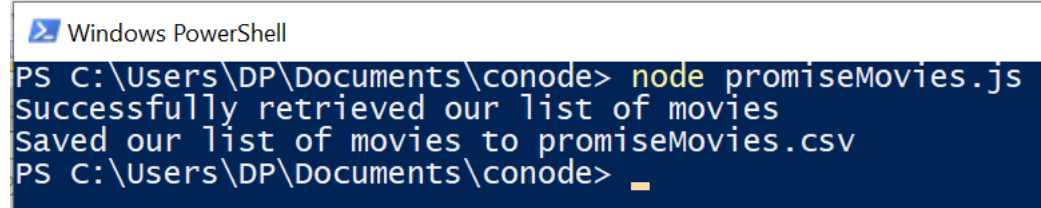
Ahora, al igual que el ejemplo anterior, vamos a exportar en un csv, para ello, debemos modificar el código a:

```
const axios = require('axios');
const fs = require('fs').promises;

axios.get('https://ghibliapi.herokuapp.com/films')
  .then((response) => {
    console.log('Successfully retrieved our list of movies');
    let movieList = '';
    response.data.forEach(movie => {
      movieList += `${movie['title']}, ${movie['release_date']}\n`;
    });

    return fs.writeFile('promiseMovies.csv', movieList);
  })
  .then(() => {
    console.log('Saved our list of movies to promiseMovies.csv');
  })
})
```

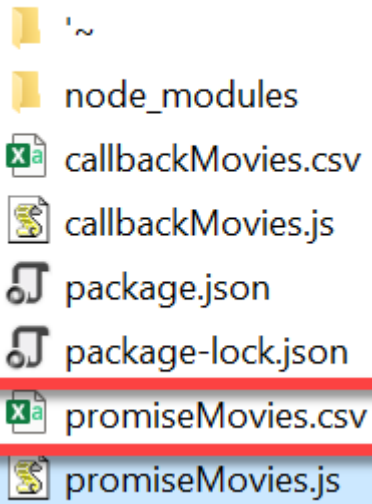
Lo que nos resultaría en:



```
Windows PowerShell
PS C:\Users\DP\Documents\conode> node promiseMovies.js
Successfully retrieved our list of movies
Saved our list of movies to promiseMovies.csv
PS C:\Users\DP\Documents\conode> █
```





Nos dice que se creó el fichero csv, vamos a la carpeta y efectivamente ahí esta:

Nombre






- ~
- node_modules
- callbackMovies.csv
- callbackMovies.js
- package.json
- package-lock.json
- promiseMovies.csv
- promiseMovies.js


Si lo abrimos veremos:


Autoguardado ☐    

Archivo Inicio Insertar Disposici



 

Pegar 






Portapapeles 

Calibri 11

N *K* S  

Fuente

A1    C

	A	B	C
1	Castle in the Sky, 1986		
2	Grave of the Fireflies, 1988		
3	My Neighbor Totoro, 1988		
4	Kiki's Delivery Service, 1989		
5	Only Yesterday, 1991		
6	Porco Rosso, 1992		
7	Pom Poko, 1994		
8	Whisper of the Heart, 1995		
9	Princess Mononoke, 1997		
10	My Neighbors the Yamadas, 1999		
11	Spirited Away, 2001		
12	The Cat Returns, 2002		
13	Howl's Moving Castle, 2004		
14	Tales from Earthsea, 2006		
15	Ponyo, 2008		
16	Arrietty, 2010		
17	From Up on Poppy Hill, 2011		
18	The Wind Rises, 2013		
19	The Tale of the Princess Kaguya, 2013		
20	When Marnie Was There, 2014		
21	The Red Turtle, 2016		

RECURSOS COMPLEMENTARIOS		
Recurso	Título	Cita referencial
Web	Manejo s de archivos y asincro nismo en NodeJS	https://utn-fullstack.github.io/clases/clase1.html
Web	Funcion es asíncro nas	https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/AsyncFunction

ACTIVIDAD DE EVALUACIÓN DE LA SEMANA	
Nombre de la Actividad	Probando asincronismo con nodejs
Tipo de Actividad	Tarea
Tipo de Participación	Individual
Instrucciones para la actividad	<ol style="list-style-type: none"> Usando asincronismo en nodejs, consulta el valor del bitcoins. Aquí te compartimos la url de la api rest donde puedes consultar: https://api.coindesk.com/v1/bpi/currentprice.json Resolver ejercicios planteados por docente durante clase.
Fecha de Entrega	Durante clase práctica
Criterios de Evaluación	Funcionalidad: 50% Autoría: 20% Github:30%
Ponderación	50% Laboratorio I