
SENPAI
ACADEMY

Node.js

Asynchronous

SENPAI
ACADEMY

Node.js - Asynchronous

Node.js es una librería asíncrona por definición, está basado en eventos de entrada y salida (E/S), por lo que usa constantemente los llamados ***callbacks***

En un programa sincrónico podríamos hacer algo así:

```
1  // Seudocodigo
2
3  function procesarData () {
4    var data = fetchData ();
5    data += 1;
6    return data;
7  }
```

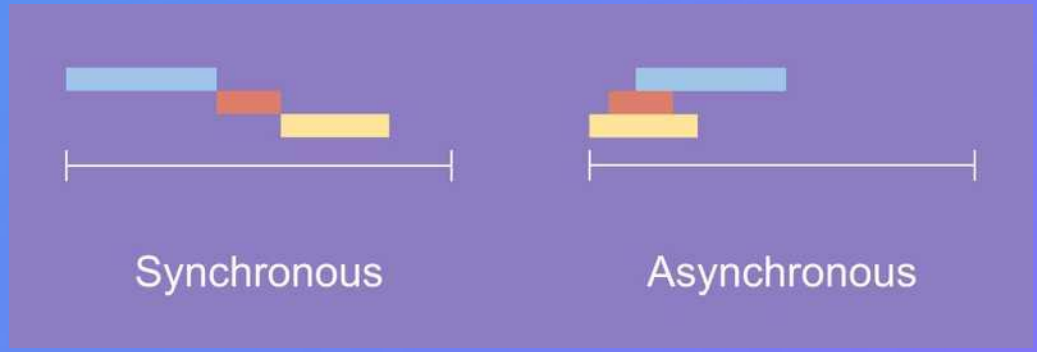
Lo mismo pero en Node.js se vería de esta forma:

```
1  function processData (callback) {
2    fetchData(function (err, data) {
3      if (err) {
4        console.log("Ocurrió un error");
5        return callback(err);
6      }
7      data += 1;
8      callback(data);
9    });
10 }
```

SENPAI ACADEMY

Node.js - Asynchronous

milliseconds	Blocking API	Non-Blocking API
20	Begin Request 1	Begin Request 1
40		Begin Request 2
60	*waiting on database...*	Begin Request 3
80		Begin Request 4
100	**End Request 1**	**End Request 1**
120	Begin Request 2	**End Request 2**
140		**End Request 3**
160	waiting on database...	**End Request 4**
180		Begin Request 5
200	**End Request 2**	Begin Request 6
220	Begin Request 3	Begin Request 7
240		Begin Request 8
260	waiting on database...	**End Request 5**
280		**End Request 6**
300	**End Request 3**	**End Request 7**
320	Begin Request 4	**End Request 8**
340		Begin Request 9
360	waiting on database...	Begin Request 10
380		Begin Request 11
400	**End Request 4**	Begin Request 12
420	Begin Request 5	**End Request 9**
440		**End Request 10**
460	waiting on database...	**End Request 11**
480		**End Request 12**
500	**End Request 5**	Begin Request 13



SENPAI ACADEMY

Node.js - Callbacks

Los ***callbacks*** son funciones que le damos a Node.js para que ejecute al finalizar determinada operación (que puede llevar mucho o poco tiempo).

Esto nos permite tener todas las operaciones de E/S que el sistema operativo soporte ocurriendo a la misma vez.

Por ejemplo, en un servidor web con cientos o miles de requests pendientes con múltiples consultas bloqueadas, nos permite seguir respondiendo a las nuevas requests, y a medida que se resuelvan las bloqueadas, responder esas también.

SENPAI
ACADEMY

Node.js - Callbacks

```
1  function asyncOperation ( a, b, c, callback ) {
2    // ... lista de sentencias ...
3    if ( /* ocurre un error */ ) {
4      return callback(new Error("Ocurrió un error"));
5    }
6    // ... mas trabajo ...
7    callback(null, d, e, f);
8  }
9
10 asyncOperation ( params.. , function ( err, returnValues.. ) {
11   // Este código se ejecuta luego de que la operación asincrónica termine
12 });
```

Está es la típica estructura de las funciones asincrónicas que crearemos y utilizaremos. Esto es siguiendo la convención de "Error callback".

- La función de callback será el último parámetro que le vamos a dar una función asincrónica.
- El primer parámetro que recibe el callback es el error (si es que ocurrió alguno).
- Si no ocurrió ningún error, el callback será invocado con el primer parámetro en null (el error), y los resultados en los demás parámetros.

SENPAI ACADEMY

Node.js - Callbacks

```js
/*
 * Senpai NodeJS - Ejemplo Async Code
 */
const fs = require('fs');

console.log('Comienza nuestro archivo');

// Lectura de archivo asincrónica
fs.readFile('resource.json', 'utf-8', (err, data) => {
  if (err) throw err;
  console.log('Letura de archivo con éxito');
  console.log(data);
 });

// Escritura de archivo asincrónica
fs.writeFile('message.txt', 'Hello Node.js', (err) => {
  if (err) throw err;
  console.log('Escritura de archivo con éxito');
});

// Creación asincrónica de una carpeta
fs.mkdir('./tmp/folder', { recursive: true }, (err) => {
  if (err) throw err;
  console.log('Creación de carpeta exitosa');
});

console.log('Termina nuestro archivo');
```

Las operaciones de lectura o escritura son clásicos ejemplos de E/S, así como también las llamadas a obtener información por la Red.

En este caso, la librería ***fs*** (que viene integrada con Node.js, documentación) nos provee métodos asincrónicos para leer, escribir, o crear carpetas, y siempre recibe como último parámetro una función *callback* para ejecutar cuando termine la operación.

¿Cuál será el orden de los console.log en la terminal?

SENPAI ACADEMY

Node.js - Callbacks

```
1  // En main.js de lista-de-tareas
2
3  const botonAgregar = document.getElementById('agregar');
4
5  botonAgregar.addEventListener('click', function () {
6    crearNuevaTarea(barrita.value, listaDeBoton.value);
7  });
```

Los callbacks nos permiten reaccionar a eventos, es decir, escribir código que queremos que se ejecute cuando ocurra algo “en el futuro”, osea, reaccionar a eventos.

Esto ya lo han visto en Frontend, cuando agregamos un event listener por ejemplo.

SENPAI ACADEMY

Event Loop

SENPAI
ACADEMY

Node.js - Event Loop

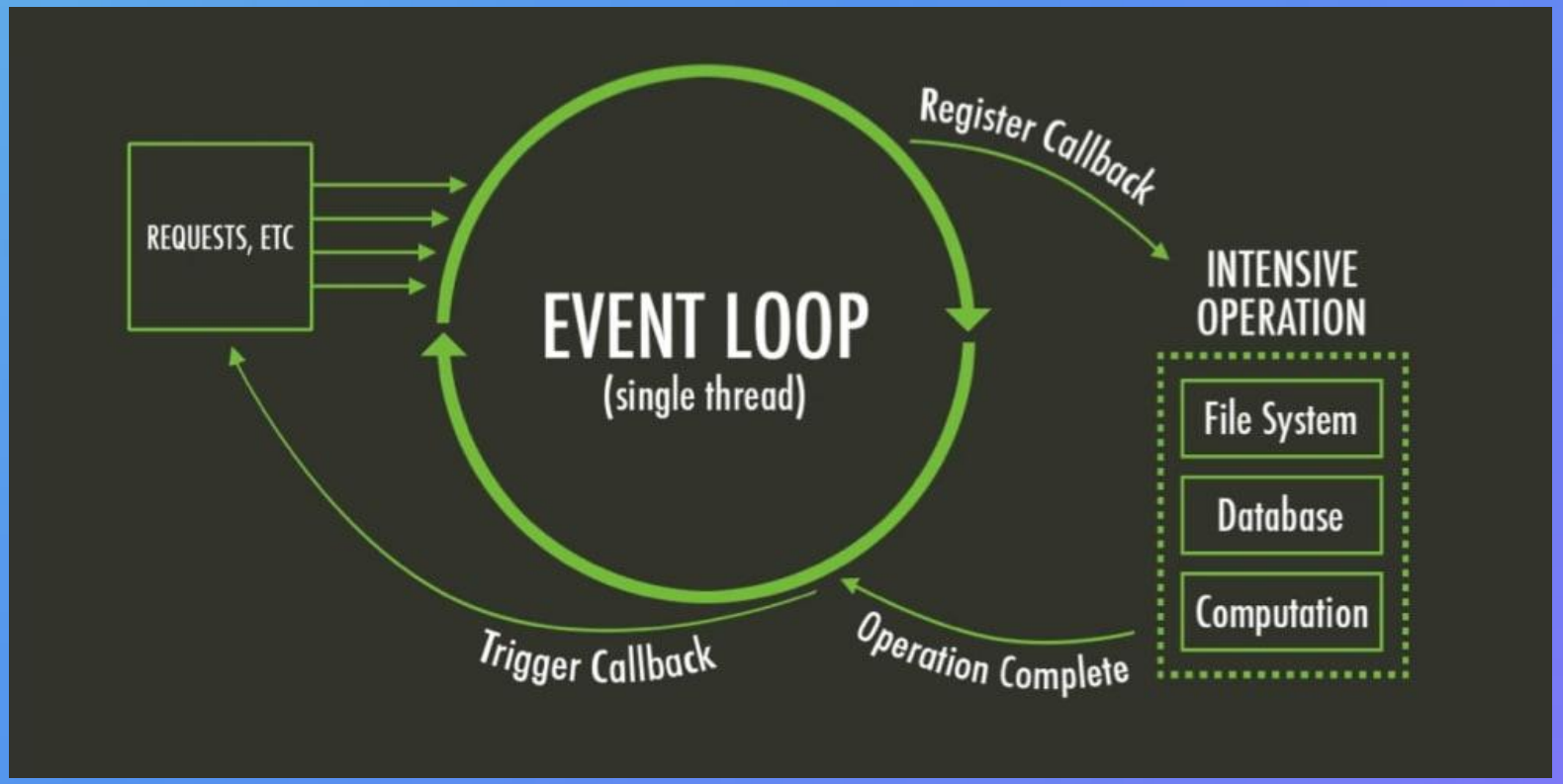
El “Event Loop” o “Bucle de Eventos” es lo que permite que Node.js sea asíncrono, y que no se bloquee con operaciones de E/S.

El código JavaScript corre en un hilo único (conocido como “single-thread”), lo que quiere decir que solo una sentencia se está ejecutando cada vez.

Esta limitación es en realidad muy útil, ya que nos ahorramos los problemas de concurrencia que surgen cuando programamos en entornos “multi-thread”.

SENPAI
ACADEMY

Node.js - Event Loop

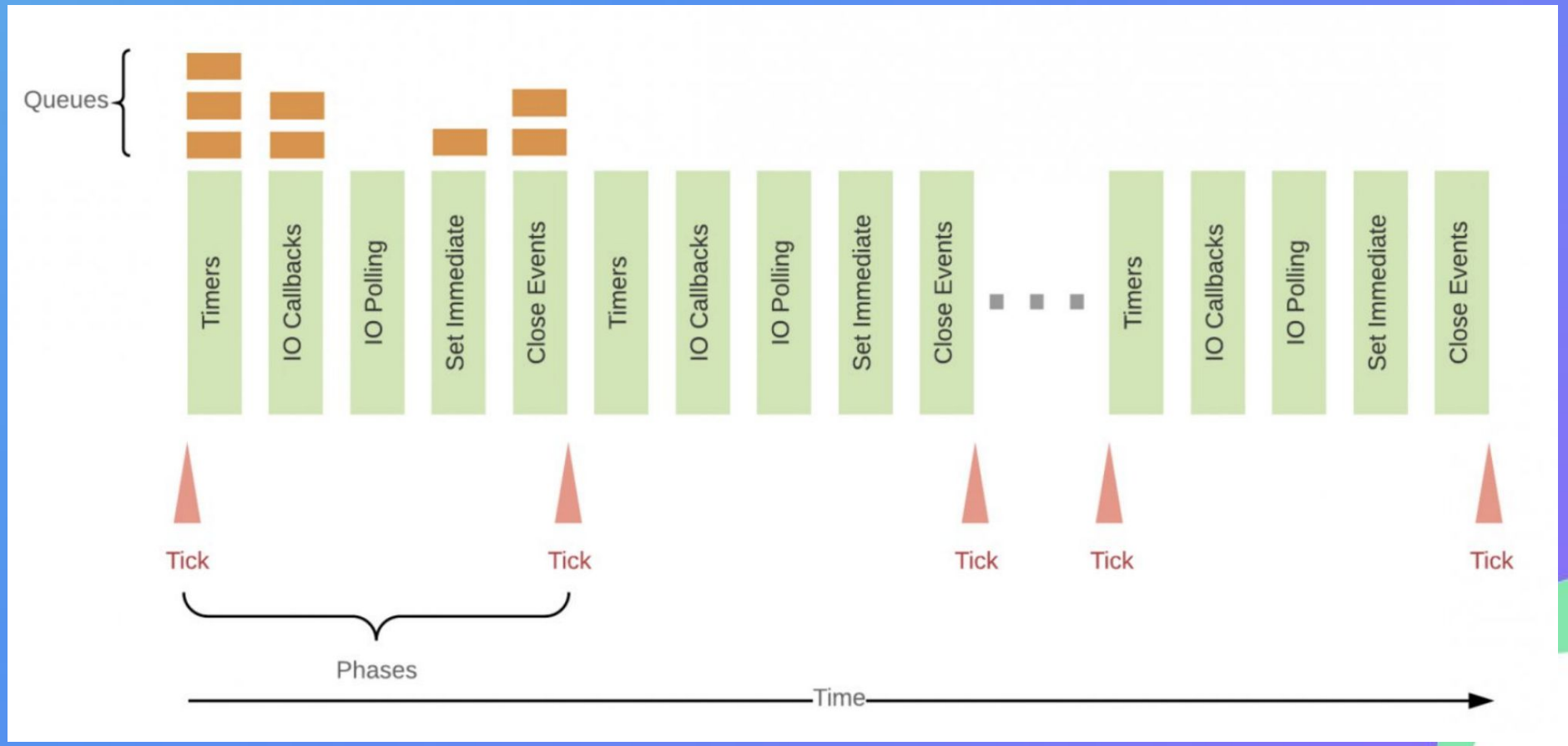


SENPAI ACADEMY

Node.js - Event Loop

Las fases del Event Loop son:

- Timers
- IO Callbacks
- IO Polling
- Set Immediate
- Close Events



SENPAI ACADEMY

Node.js - Event Loop

Timers phase
Todo lo agendado por ***setTimeout()*** o ***setInterval()*** es procesado en esta fase.

IO Callbacks phase
Se ejecutan callbacks de eventos de E/S (In Out). Como la mayoría del código que hagamos van a ser callbacks, esta es la fase donde se ejecuta la mayoría de nuestro código.

IO Polling phase
Se consulta el estado de los eventos de E/S para agendarlos para el próximo loop.

SENPAI
ACADEMY

Node.js - Event Loop

Set Immediate phase
Se ejecutan todos los callbacks registrados vía ***setImmediate()***.

Close phase
Se ejecutan todos los eventos “close”.

Más información:

- Documentación
- Charla explicando el Event Loop.

SENPAI ACADEMY

Node.js - Ejercicio 1

Escribir un pequeño programa en Node.js en ***app.js*** que liste todos los archivos de la carpeta donde se ejecuta.

- Utilizar la librería ***fs*** de Node.js y el método ***readdir***
- Este método recibe como primer parámetro la ruta a leer, y como segundo parámetro una función callback que ejecutará con el resultado.
- Imprimir cada archivo a la consola.

Tip: En Node.js existe la variable global "***__dirname***" que hace referencia a donde está ubicado el archivo ejecutando.

Extra: Utilizando otra librería integrada en Node.js llamada "***path***" y el método ***path.join*** leer el contenido de otra carpeta dentro de donde esté el ***app.js***.

SENPAI ACADEMY

Node.js - Ejercicio 2

Escribir un pequeño programa en Node.js en ***app2.js*** que nos pregunte en la consola nuestro nombre, luego nuestro país de origen, e imprima en consola por ejemplo: “Diego es un ciudadano de Uruguay”

Para esto precisan algunas guías:

- Tienen que requerir la librería ***readline*** integrada en Node.js (documentación)
- Luego tienen que crear una interfaz para readline con el siguiente código:

```
const rl = readline.createInterface({
 input: process.stdin,
 output: process.stdout
});
```

- Luego en el objeto ***rl*** pueden llamar al método ***question*** que recibe dos parámetros:
  - La pregunta
  - Un callback que invoca con el texto ingresado en la terminal
- Extra: recuerden que para armar el mensaje final pueden utilizar String Templates: *`El nombre es ${name}..`*

```
`El nombre es ${name}..`
```

SENPAI ACADEMY

Node.js - Ejercicio 3

Escribir un pequeño programa en Node.js en ***app3.js*** que:

- Imprima un mensaje en la consola indicando que comenzó.
- Cree un timer que 3 segundos más tarde imprima un “Hola Mundo” en la terminal, utilizando la función ***setTimeout*** (documentación).
- Imprima un mensaje final en la consola (que se debería ver antes que el mensaje en el timer)

SENPAI ACADEMY

SENPAI
ACADEMY

gustavguez

gustavguez

GUSTAVO RODRIGUEZ

FULL STACK DEVELOPER
SOLCRE