Event Loop y Asincronía

DEV.F:
DESARROLLAMOS(PERSONAS);

Código síncrono

Un código síncrono es aquel código donde cada instrucción espera a la anterior para ejecutarse.

```
1   console.log('Primero');
2   console.log('Segundo');
3   console.log('Tercero');
```

Código asíncrono

Un código asíncrono no espera a las instrucciones diferidas y continúa con su ejecución. Entiéndase por instrucción diferida cualquier cosa que implique un retraso. Esto evita que haya código bloqueante, el código bloqueante se forma en la cola.

```
1   console.log('Primero');
2   setTimeout(_ => {
3     console.log('Segundo');
4   }, 1);
5   console.log('Tercero');
```

¿Dónde aparece el código asíncrono?

- Por naturaleza de JavaScript (Lenguaje no bloqueante).
- Llamado de APIs.
- Escritura de archivos.
- Operaciones con base de datos.
- Al realizar una compra en la plataforma X, se válida primero la tarjeta del cliente y hasta que se comprueba que es válida se ejecuta la compra.
- Filtrar una tabla de datos y generar un reporte de la información.
- Envío de un correo de una sanción a un crédito.

Ventajas de la asincronía

- Permite tener una mejor respuesta en las aplicaciones y reduce el tiempo de espera del cliente.

Métodos de manejo de la asincronía

Para un desarrollador JS es fundamental aprender a buscar mecanismos para dejar claro cuándo ciertas tareas tienen que procesarse de forma síncrona (quedarse a la espera) y cuándo deben ejecutarse de forma asíncrona.

- **Callbacks:** Consiste en pasar una función como parámetro y ejecutarla en el momento que lo necesitemos.
- **Promesas:** Se basan en 3 estados. Cuándo se lanza la petición (pending) y sus posibles respuestas (resolve y reject).
- **Async await (ES8):** Función con la sugar syntaxis de es6 y simula que el código es asíncrono (por detrás sigue siendo una promesa).

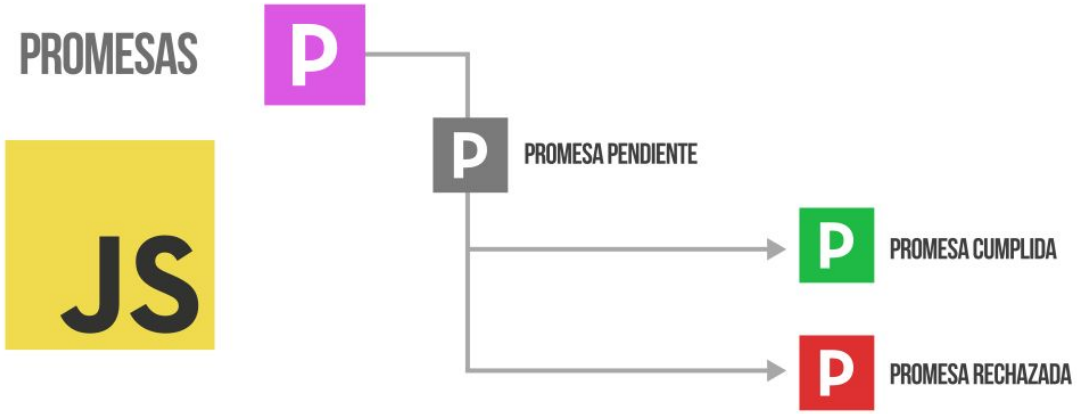
Callbacks

- Consiste en pasar una función como parámetro a otra, para que la segunda función la ejecuté cuando lo requiera. Dicho de otra manera, es la programación la ejecución del código dentro del callback.
- Solución sencilla pero estéticamente compleja de leer y caótica.

```
firstTask(data, function(err, result) {
  secondTask(data, function(err, result) {
    thirdTask(data, function(err, result) {
      fourthTask(data, function(err, result) {
        fifthTask(data, function(err, result) {
          // Code
        });
      });
    });
  });
});
```

Promises

- Consiste en crear un bloque de código y consumirlo. Al consumirlo, primero se tiene una expectativa de su respuesta, después una respuesta de éxito o error.



Async await

- Son sugar syntaxis introducido en es8 (2017) para mejorar la legibilidad de las promesas. Se abandona el modelo de encadenamiento then.
- Async vuelve a las funciones una promesa y permite usar el keyword await.
- Await espera a que se resuelvan las promesas.
- Para gestionar errores hay que agregar un bloque try - catch.

Comparativa entre Node vs JavaScript

DEV.F:
DESARROLLAMOS(PERSONAS);

JS vs Node

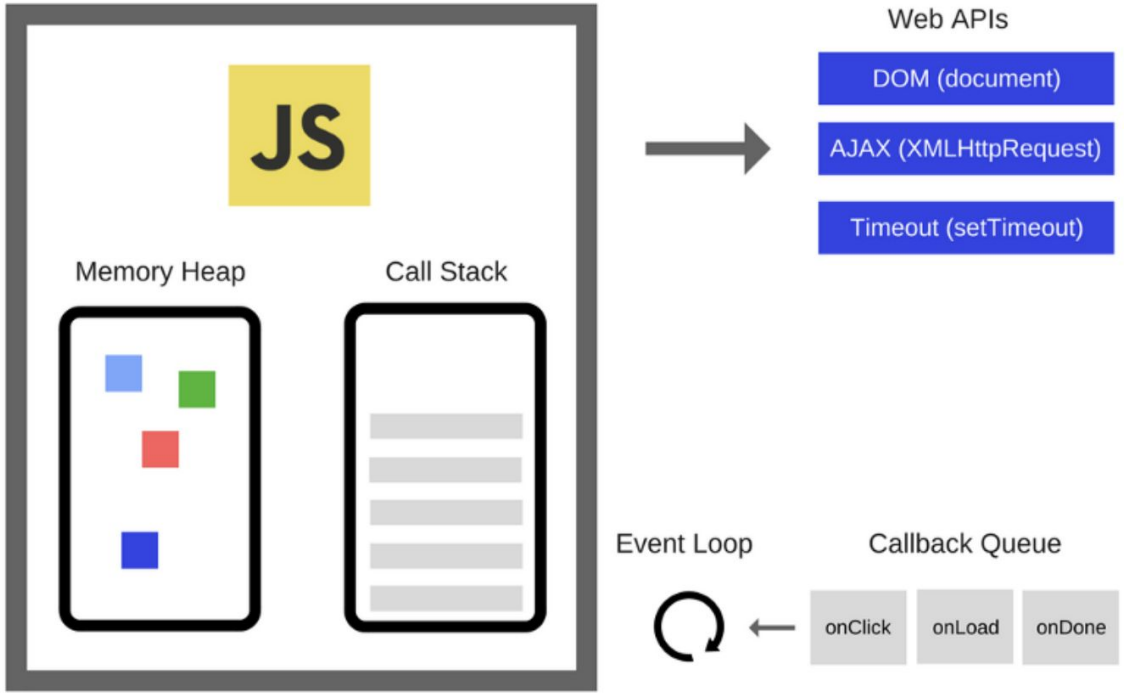
- Lenguaje de scripting vs Entorno de ejecución.
- Acceso al browser vs Acceso al servidor
- Motor del navegador vs V8.
- Js interactúa directamente con el DOM.
- Navegadores usan Libevent.
- Node usa Libuv.
- Ambos se basan en 0 retraso.
- No hay solicitudes Ajax (consumo de API’s).
- No hay temporizador (setTimeout y setInterval).

Conceptos event loop

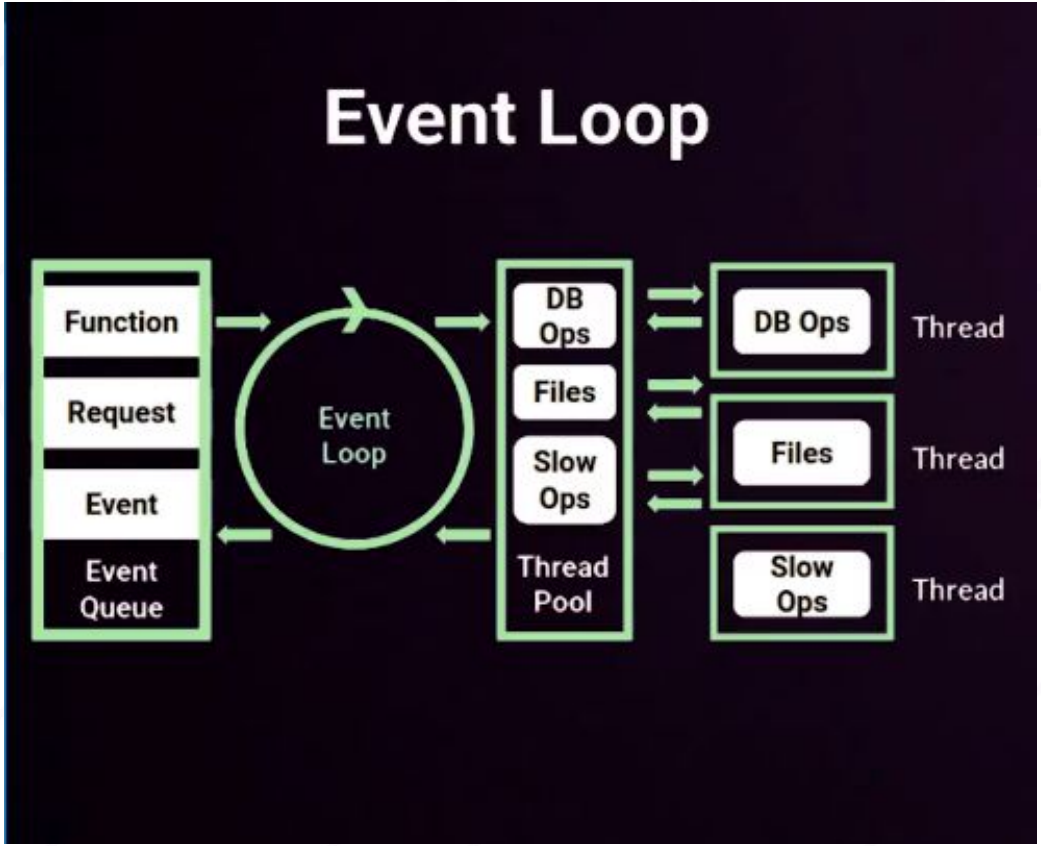
Modelo de concurrencia y ejecución de eventos. Es la base del funcionamiento de JS (y por lo tanto de node).

- Heap.
- Call stack.
- Event loop.
- API Web.
- Callback Queue.

Event Loop JavaScript (libevent)



Event Loop Node (libuv)



Ejemplo de restaurante para asincronismo

Práctica 12

- Hacer un ejemplo práctico de callbacks, promesas y async await.