

JS ASYNC

(Asincronismo en JavaScript)

DEV.F
DESARROLLAMOS(PERSONAS);

dev



OBJETIVOS

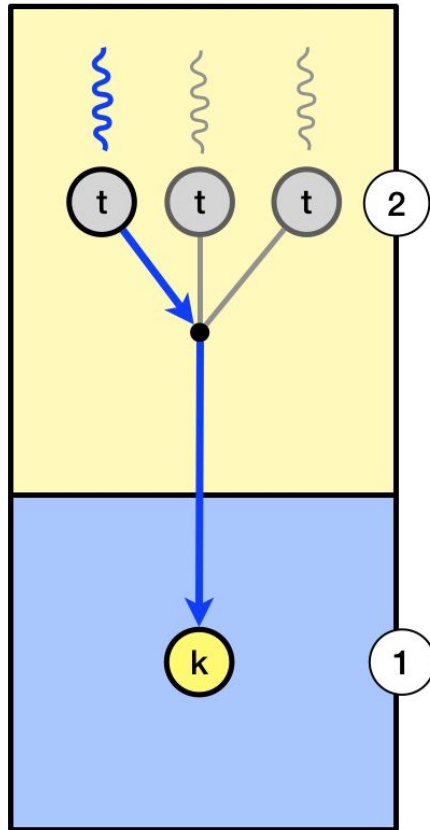
1. Entender la lógica de los hilos de ejecución.
2. Comprender la Concurrencia y Paralelismo en la ejecución de procesos y su rendimiento.
3. Aprender como JavaScript se ejecuta con ayuda del Call Stack y el Callback queue.
4. Entender qué son los Callbacks.

Hilos de Ejecución

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Hilos de ejecución



Concurrencia y Paralelismo

- **Concurrencia:** Cuando dos o más tareas progresan simultáneamente.
- **Paralelismo:** Cuando dos o más tareas se ejecutan, literalmente, a la vez, en el mismo instante de tiempo

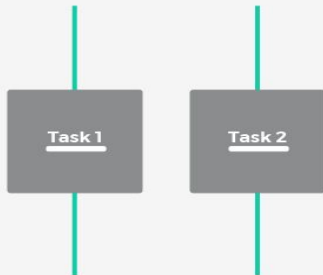
THREAD ÚNICO

MÚLTIPLES THREADS

SECUENCIAL



1

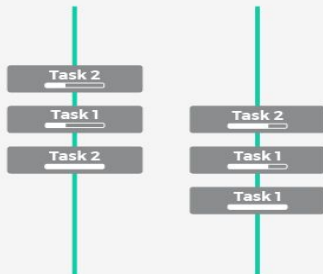


2

ENTRELAZADO

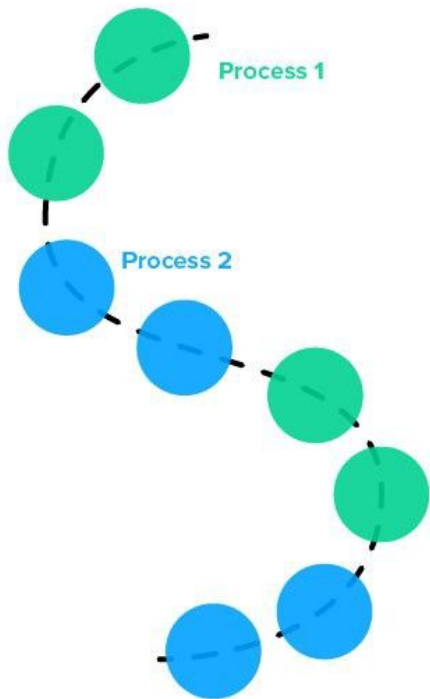


3

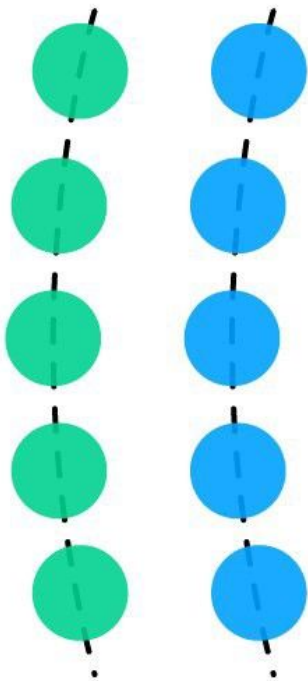


4

Concurrency



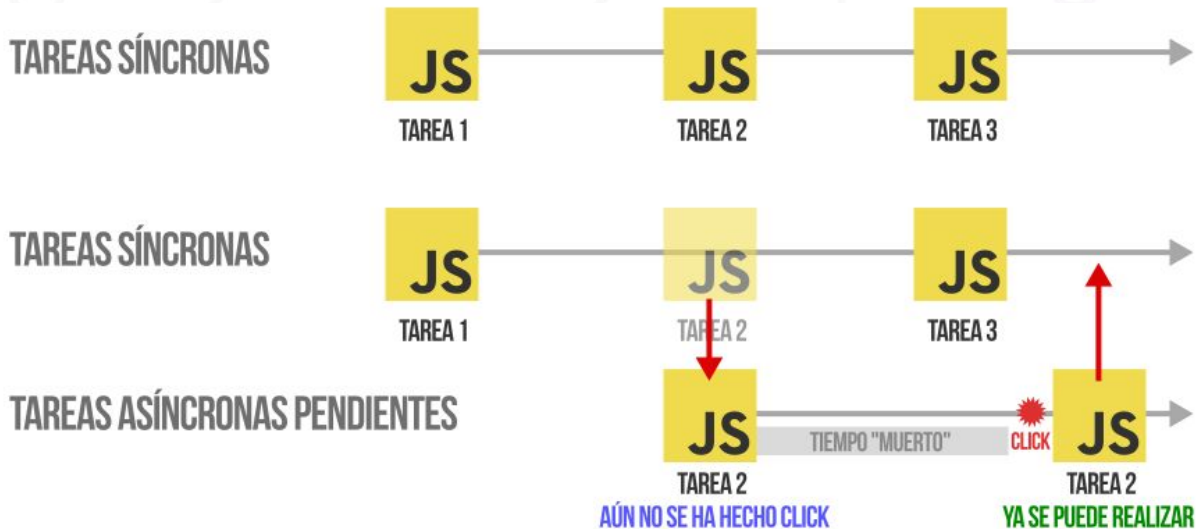
Parallelism



Concurrencia y Paralelismo

- **Concurrencia:** Capacidad del CPU para ejecutar más de un proceso al mismo tiempo (ayudándose del número de núcleos).
- **Paralelismo:** Toma un único problema y mediante concurrencia llega a la solución más rápido (divide y venceras).

Síncrono y Asíncrono



Síncrono: Toda operación o tarea se ejecuta de forma secuencial y, por tanto debemos esperar para procesar el resultado..

Asíncrono: la finalización de la operación es notificada al programa principal. El procesado de la respuesta se hará en algún momento futuro.

Callbacks

DEV.F
DESARROLLAMOS(PERSONAS);

dev



```
function saludar(nombre) {  
  alert('Hola ' + nombre);  
}  
  
function procesarEntradaUsuario(callback) {  
  var nombre = prompt('Por favor ingresa tu nombre.');
```


 callback(nombre);
}

procesarEntradaUsuario(saludar);

¿Que es un Callback? (llamada de vuelta)

- Es una función que recibe como parámetro otra función y la ejecuta.
- La función “callback” por lo regular va a realizar algo con los resultados de la función que la está ejecutando.
- Es una forma de ejecutar código de forma “asíncrona” ya que una función va a llamar a otra.
- Cuando pasamos un callback solo pasamos la definición de la función y no la ejecutamos en el parámetro. Así, la función contenedora elige cuándo ejecutar el callback.

Ejemplo de casos de uso de Callbacks

```
function successCallback() {  
  // Do stuff before send  
}
```

```
function successCallback() {  
  // Do stuff if success message received  
}
```

```
function completeCallback() {  
  // Do stuff upon completion  
}
```

```
function errorCallback() {  
  // Do stuff if error received  
}
```

Ejemplo de Callback

```
/* Tenemos 2 funciones que devuelven un valor */  
function soyCien() { return 100; }  
function SoyDoscientos() { return 200; }  
  
/* Esta función recibe como parametro 2 funciones y las ejecuta */  
function sumaDosFunciones(functionOne, functionTwo) {  
    const suma = functionOne() + functionTwo();  
    return suma; // retornando un nuevo valor, en este caso su suma  
}  
  
/* Invocamos a sumaDosFunciones y le pasamos 2 funciones como parámetros */  
console.log(sumaDosFunciones(soyCien, SoyDoscientos));  
// Resultado → 300
```

Ejemplo setTimeout



```
setTimeout(function() {  
  console.log("He ejecutado la función");  
}, 2000);
```

Le decimos a `setTimeout()` que ejecute la función callback que le hemos pasado por primer parámetro cuando transcurran 2000 milisegundos (es decir, 2 segundos, que le indicamos como segundo parámetro).

Ventajas y desventajas de los callbacks

Ventajas

- Son fáciles de usar.
- Pueden solucionar problemas de flujo de una aplicación
- Ayudan a manejar excepciones.
- Son útiles cuando quieres hacer consultas a una BD o servicio web

Desventajas

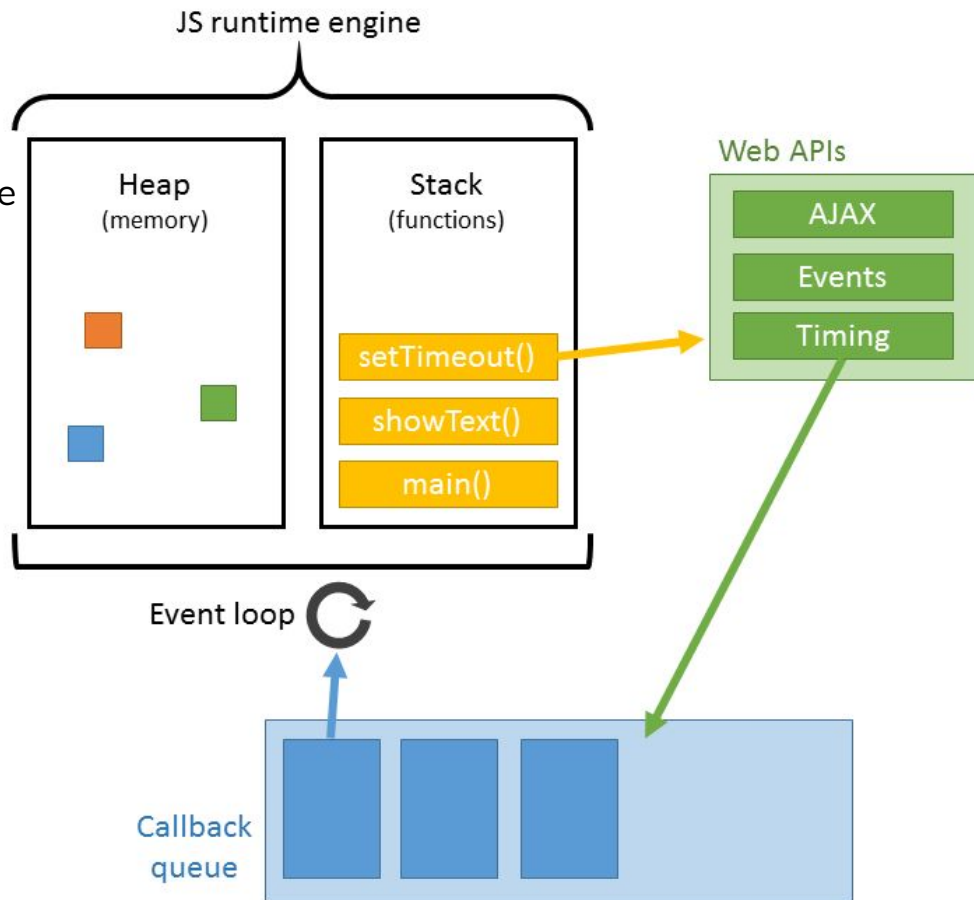
- A Veces el concepto es confuso
- Si se usa demasiado se puede caer en algo denominado “callback hell”
- El uso excesivo puede afectar el performance.
- Para programadores novatos no es muy fácil leer y entender qué hacen las funciones callback.

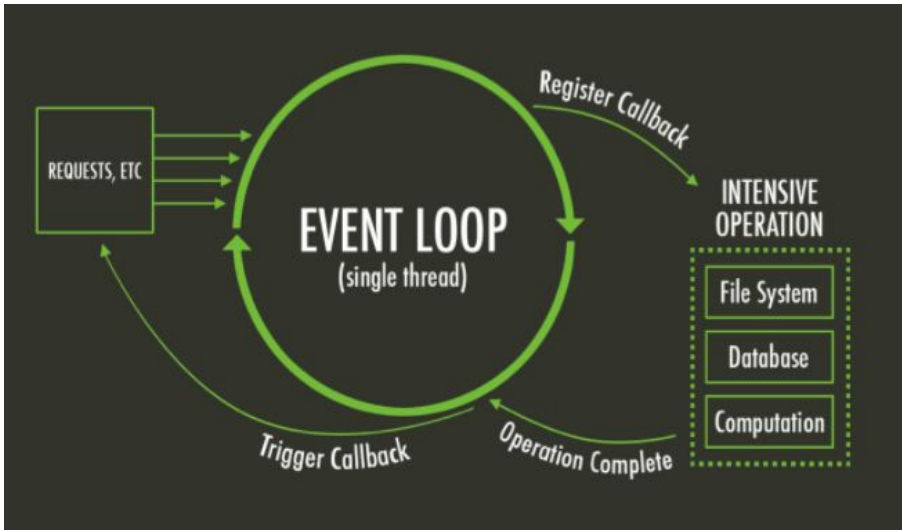
Event Loop de JavaScript

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Javascript se considera un lenguaje single thread, asíncrono y no bloqueante.



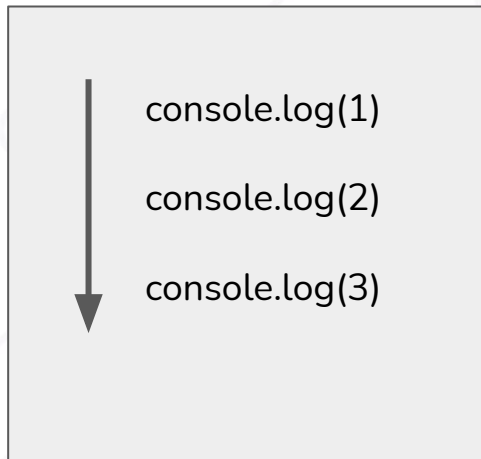


Event Loop

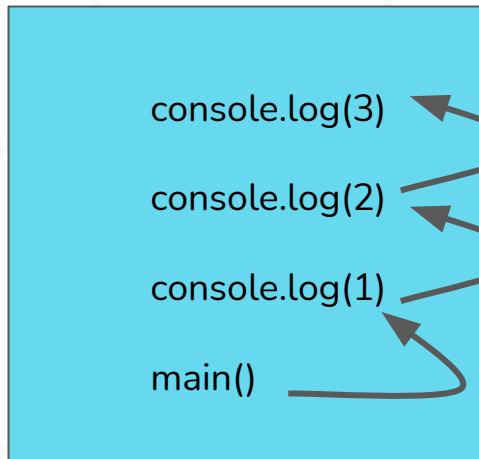
- JavaScript posee un modelo de concurrencia basado en un "loop de eventos".
- Es el motor.
- Está al pendiente de que elementos se pasan a la cola o a la pila de ejecución.
- Es el encargado de entender el orden de ejecución.
- Nunca interrumpe otros programas en ejecución. por ejemplo, puede esperar el resultado de una consulta a base de datos y seguir procesando interacciones del usuario (clicks)

¿Como funciona? - Flujo normal síncrono

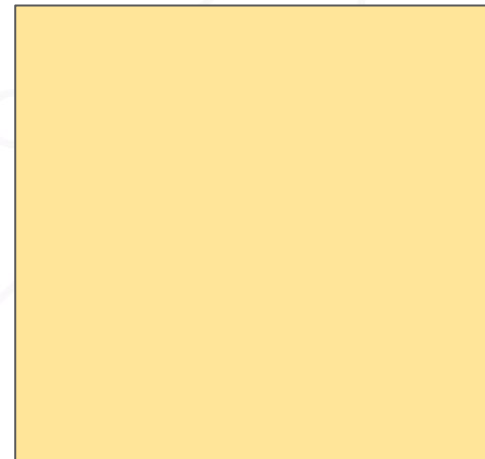
main.js



Pila de Ejecución
(call stack)



Cola de Ejecución
(callback queue)



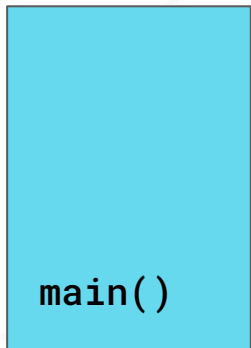
Terminal



Normalmente en JS un proceso va a la **Pila de Ejecución** y se ejecuta siguiendo el proceso LIFO (last-in-first-out)

ilustrando la ejecución en el Call Stack

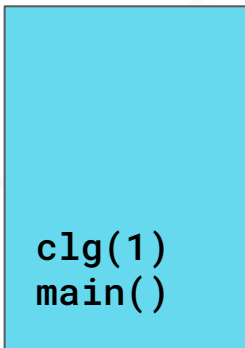
Pila de
Ejecución
(call stack)



Terminal



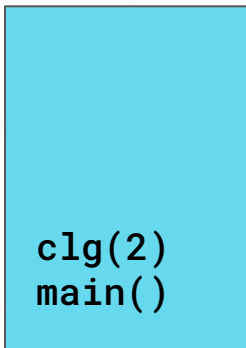
Pila de
Ejecución
(call stack)



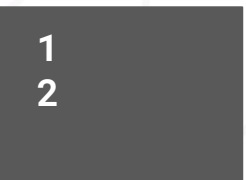
Terminal



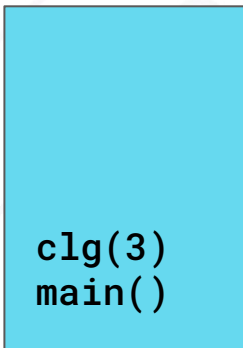
Pila de
Ejecución
(call stack)



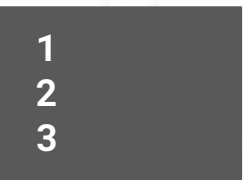
Terminal



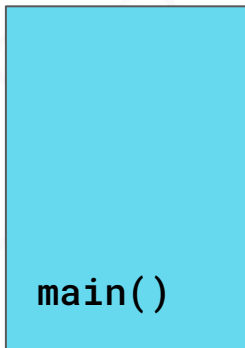
Pila de
Ejecución
(call stack)



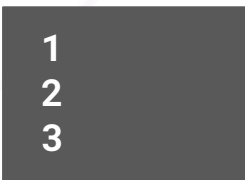
Terminal



Pila de
Ejecución
(call stack)



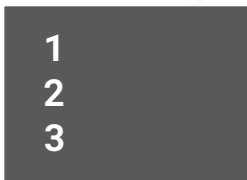
Terminal



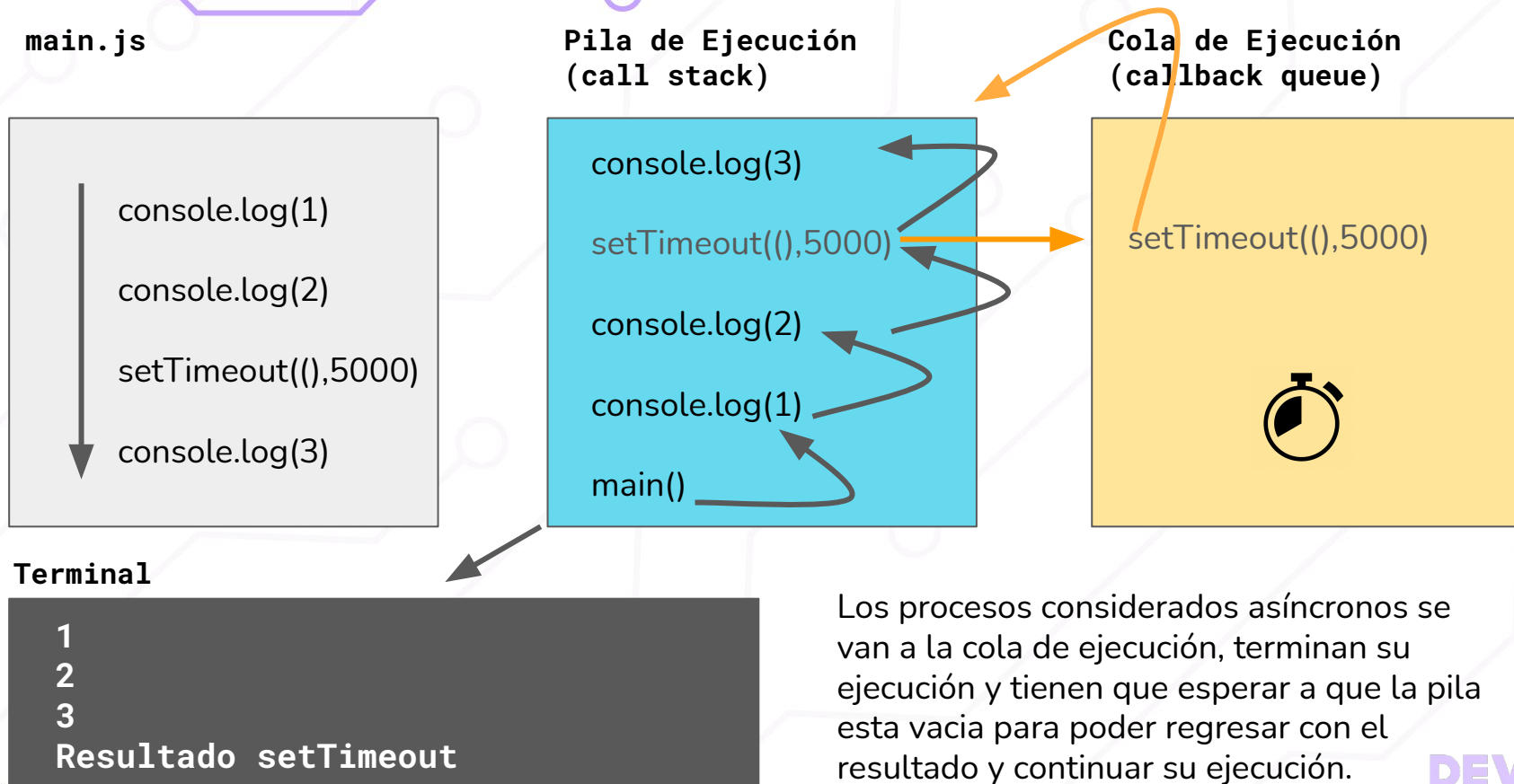
Pila de
Ejecución
(call stack)



Terminal



¿Como funciona? - Flujo asíncrono



Los procesos considerados asíncronos se van a la cola de ejecución, terminan su ejecución y tienen que esperar a que la pila esta vacia para poder regresar con el resultado y continuar su ejecución.

Ejemplo #1: ¿Cuál es el resultado de este código?



```
console.log("---Todo en Pila de Ejecución---");  
console.log(1);  
console.log(2);  
console.log(3);
```

Ejemplo #2: ¿Cuál es el resultado de este código?



```
console.log("---El 2 y 3 van a la Cola de Ejecución---");  
console.log(1);  
// SetTimeout Espera N segundos para ejecutar un CALLBACK.  
// Recibe 2 parametros: setTimeout(callback, milisegundos)  
setTimeout(()⇒{ //Simular Ir a Base de Datos con un callback;  
    return console.log(2)  
},3000);  
setTimeout(()⇒{  
    return console.log(3)  
},2000);  
console.log(4);
```

Ejemplo #3: ¿Cuál es el resultado de este código?



```
console.log("---Simulación de Cuello de Botella---");  
console.log(1);  
setTimeout(()⇒{  
    return console.log(2);  
},2000);  
for (let index = 0; index < 9999999999; index++);  
console.log(3);
```

The logo consists of the text 'DEV.F.' in a bold, white, sans-serif font. The 'F' is stylized with three small squares at its base. The logo is centered within a dark blue diamond shape.

DEV.F.

IMPORTANTE:

Muchos de los procesos que involucran pedir información de forma externa suelen ser asíncronos.

*Por ejemplo, **las consultas a bases de datos son por naturaleza asíncronas.***