JS ASYNC

(Asincronismo en JavaScript)

DEV.F::

DESARROLLAMOS(PERSONAS);

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.
5. Usar Callbacks.

DEV.F::

ASINCRONISMO EN JS

Que es asincronismo?

Si hablamos de Asincronía hacemos referencia al suceso que NO tiene lugar en total correspondencia temporal con otro suceso.

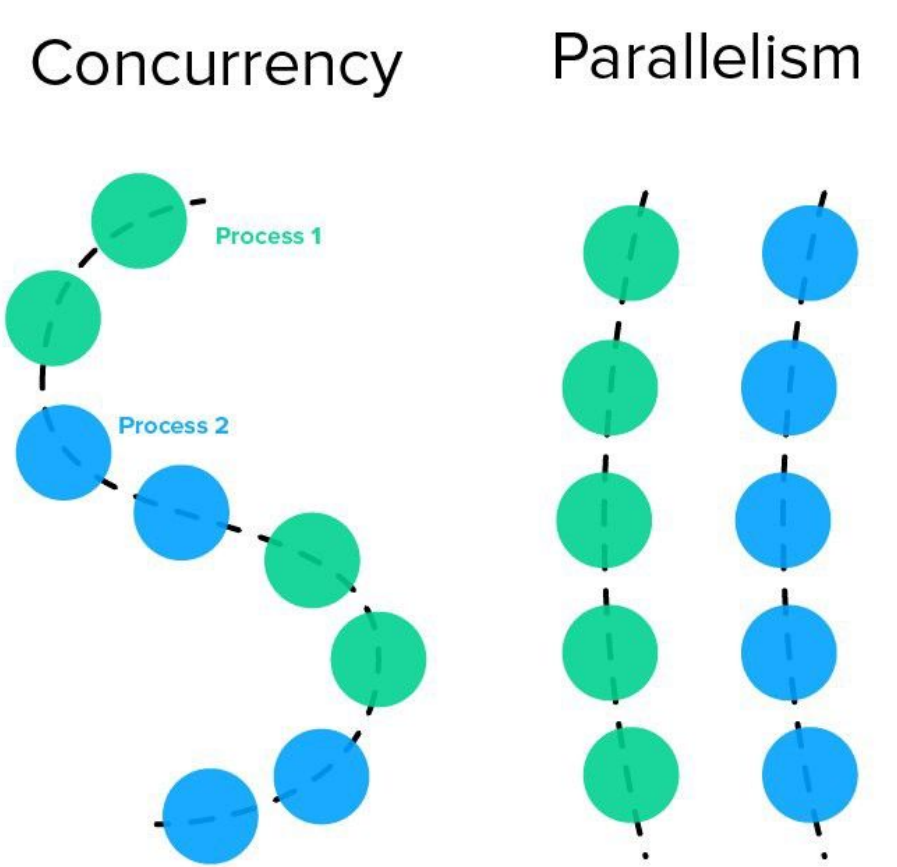
ASINCRONISMO EN JS

Que es paralelismo?

Múltiples tareas de forma simultánea.

Que es concurrencia?

Se hace una tarea después de la otra



Concurrencia y Paralelismo

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

DEV.F::

Bloqueante y No Bloqueante:

Una tarea no devuelve el control hasta que se ha completado.

Una tarea devuelve inmediatamente con independencia del resultado...
Si completo devuelve los datos, si no, un error

ASINCRONISMO EN JS

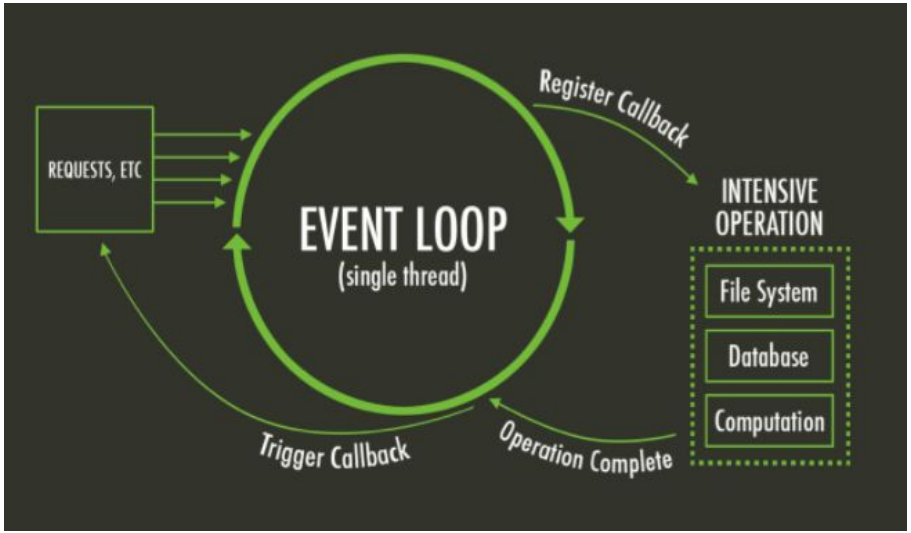
Y sincronismo?

Las tareas se ejecutan de forma secuencial, se debe esperar a que se complete para continuar con la siguiente tarea. (python y php)

Entonces... En el asincronismo:

Las tareas pueden ser realizadas más tarde, lo que hace posible que una respuesta sea procesada en diferido.

El bucle de eventos es un patrón de diseño que espera y distribuye eventos o mensajes en un programa.



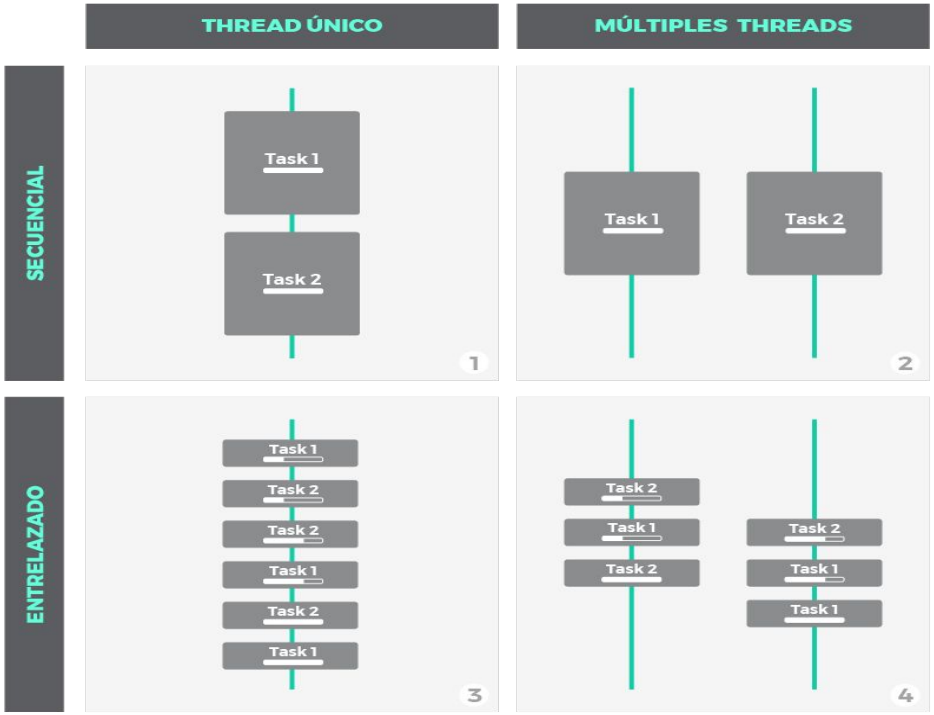
Event Loop / Bucle de eventos

- JavaScript poseé 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 (clics)

DEV.F::

Hilos de Ejecución

DEV.F::
DESARROLLAMOS(PERSONAS);



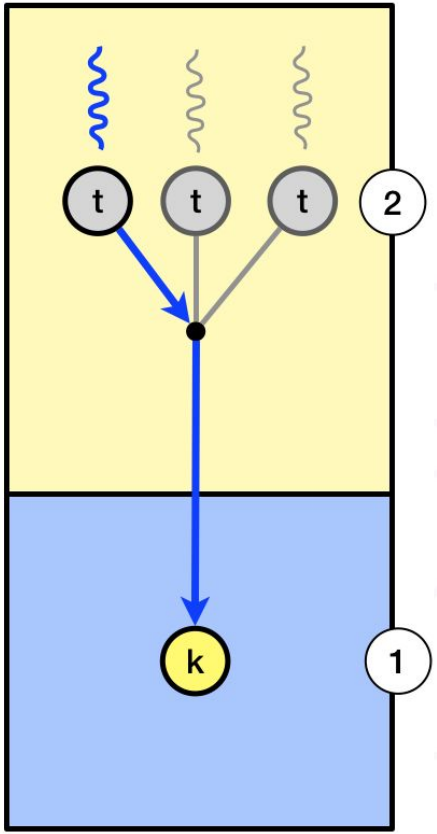
Concurrencia y Paralelismo

- **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

DEV.F::

Hilos de ejecución

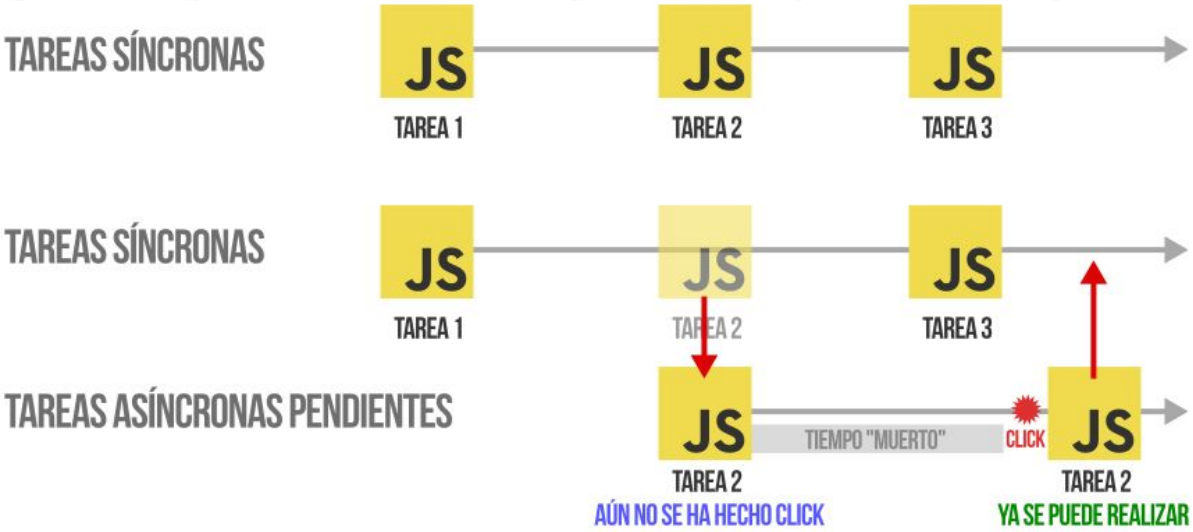


JAVASCRIPT ES:

Asíncrono y **no bloqueante**, con un **bucle de eventos** implementado con un **único hilo** para sus interfaces de I/O.

DEV.F::
DESARROLLAMOS(PERSONAS);

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.

DEV.F::

FORMAS DE MANEJAR EL ASINCRONISMO
Callbacks, Promises, Async/await

DEV.F::
DESARROLLAMOS(PERSONAS);


```
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.

DEV.F::

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 faciles 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

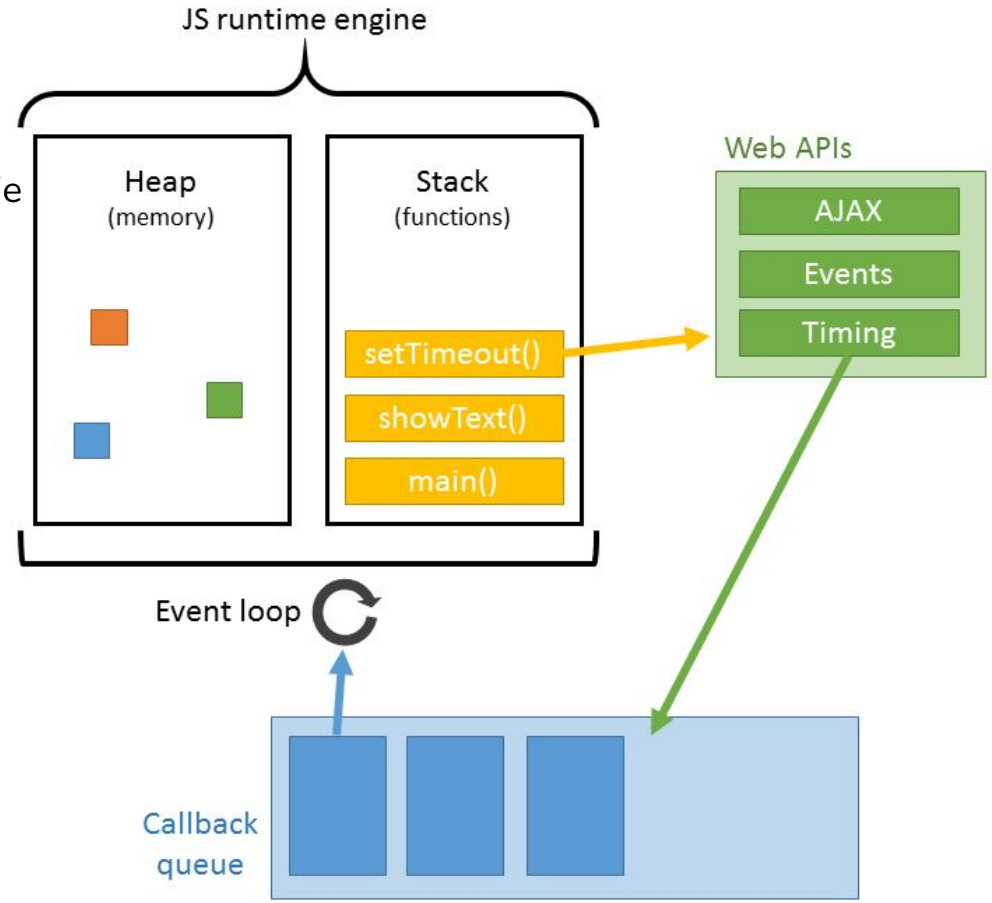
Desventajas

- 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.

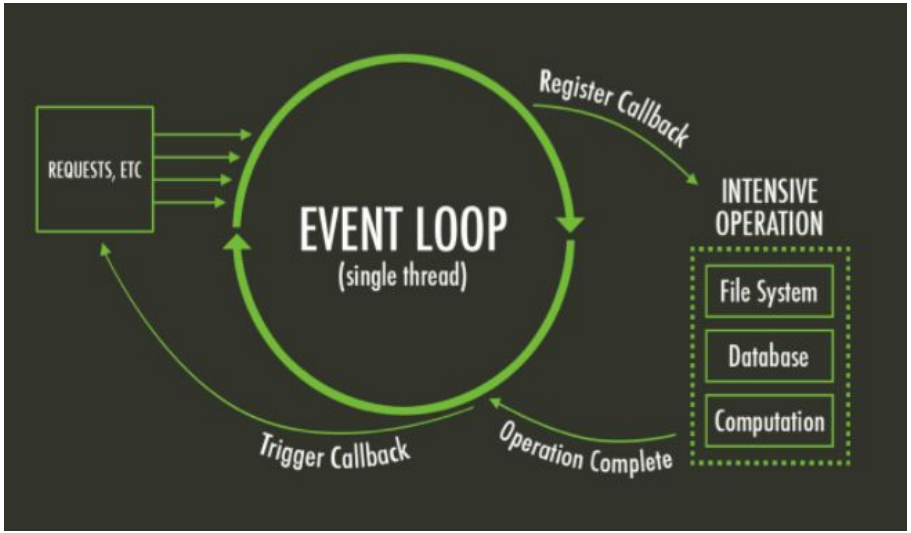
Event Loop de JavaScript

DEV.F::
DESARROLLAMOS(PERSONAS);

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



El bucle de eventos es un patrón de diseño que espera y distribuye eventos o mensajes en un programa.

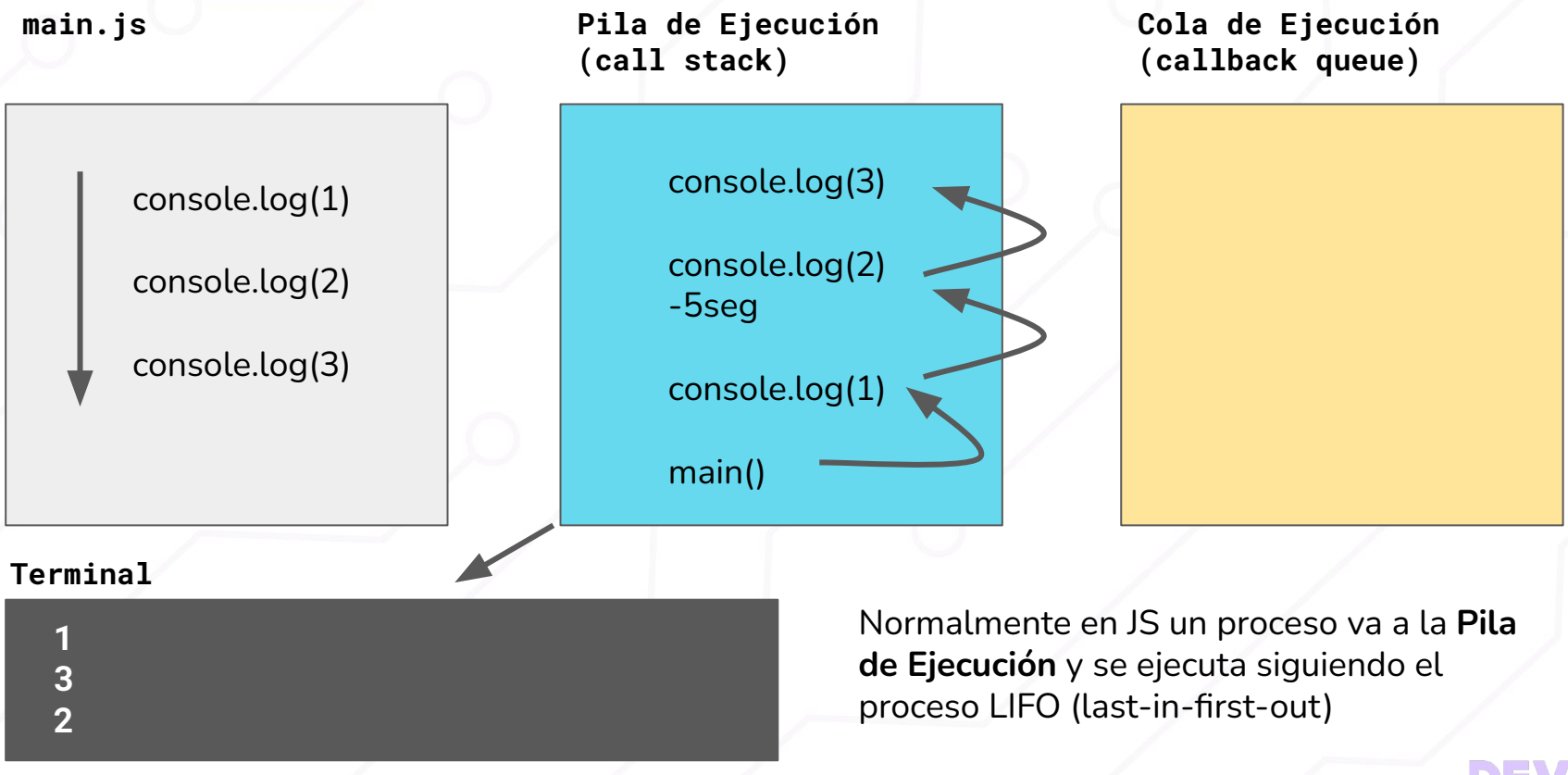


Event Loop / Bucle de eventos

- JavaScript poseé 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 (clics)

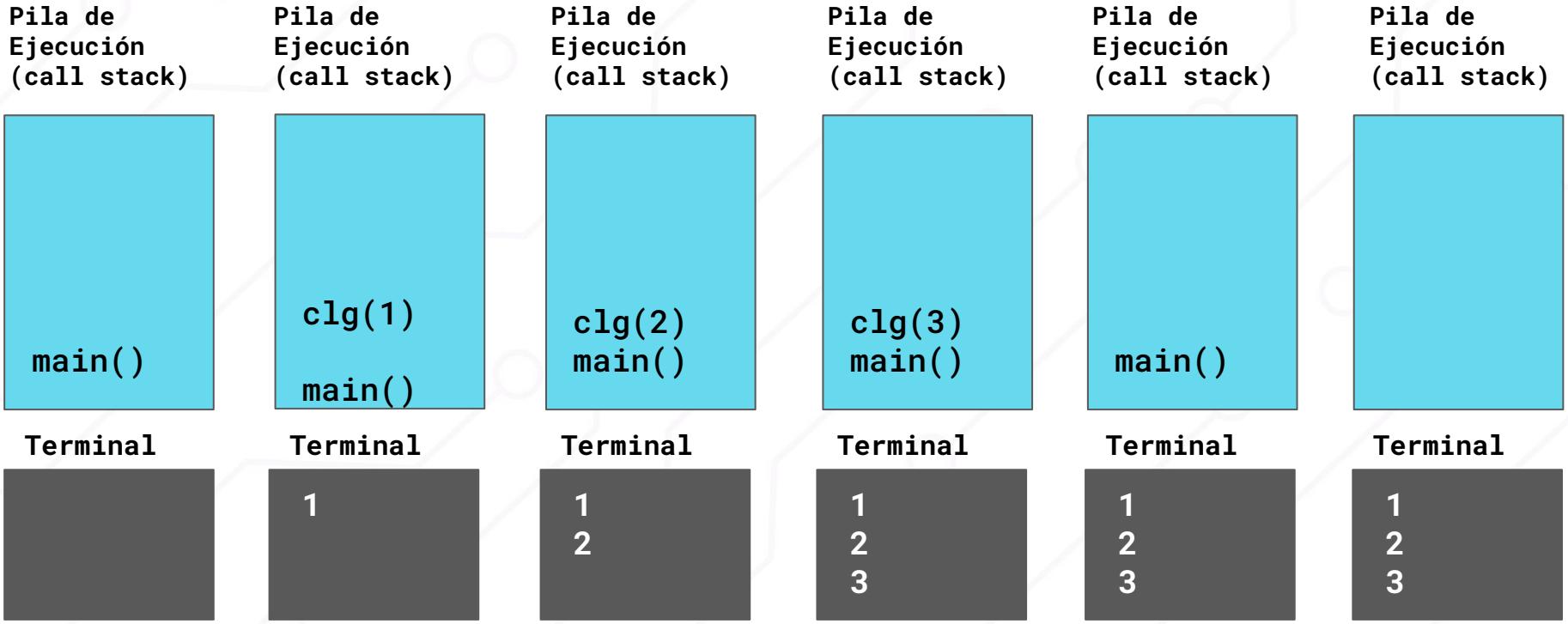
DEV.F::

¿Como funciona? - Flujo normal síncrono



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

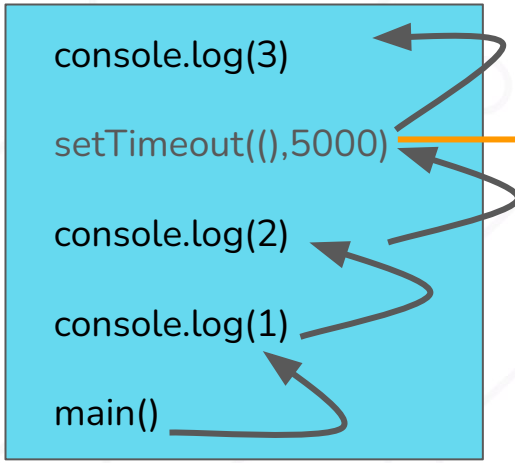


¿Como funciona? - Flujo asíncrono

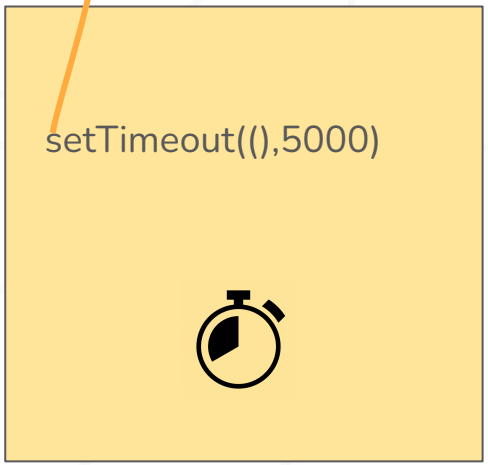
main.js

```
console.log(1)
console.log(2)
setTimeout(()),5000)
console.log(3)
```

Pila de Ejecución (call stack)



Cola de Ejecución (callback queue)



Terminal

```
1
2
3
Resultado setTimeout
```

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.

DEV.F

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);
```

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.***