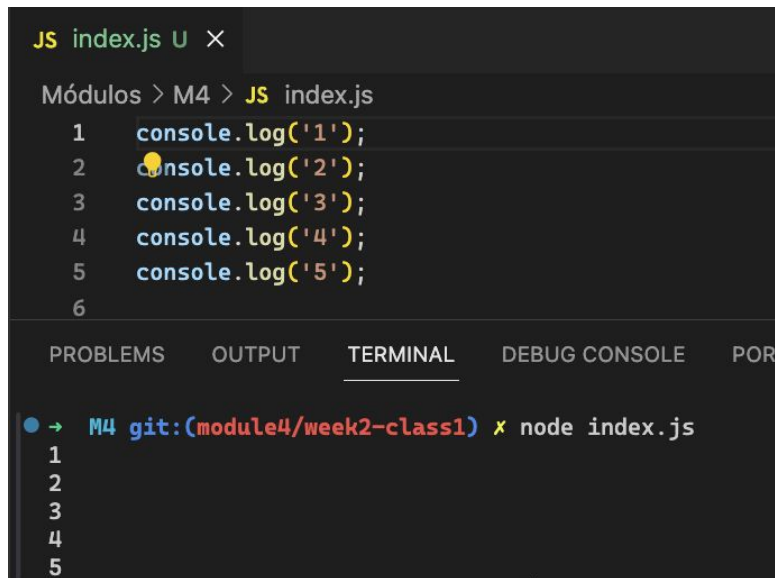


# Event Loop de JavaScript

**DEV.F**  
DESARROLLAMOS(PERSONAS);

dev

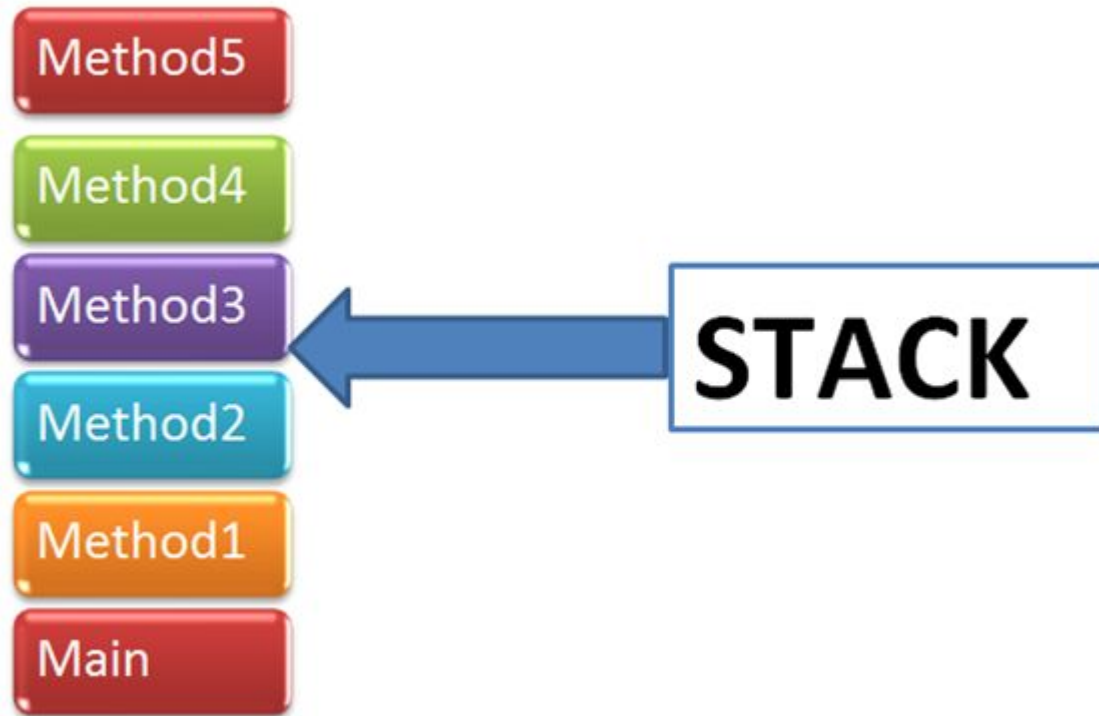


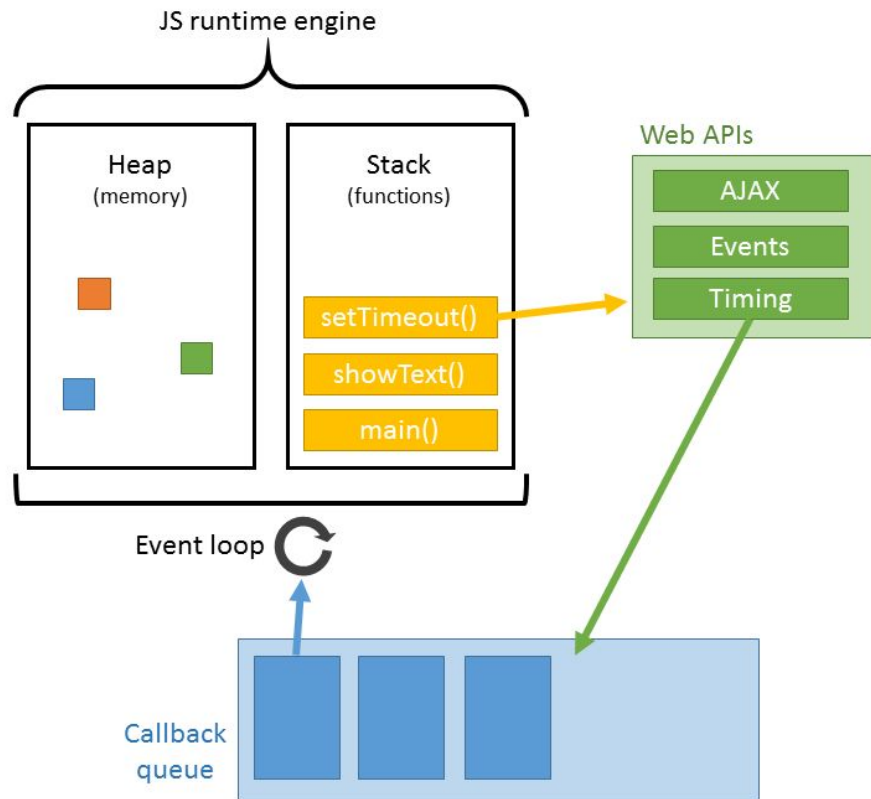
```
JS index.js U X
Módulos > M4 > JS index.js
1 console.log('1');
2 console.log('2');
3 console.log('3');
4 console.log('4');
5 console.log('5');
6

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE POR
M4 git:(module4/week2-class1) x node index.js
1
2
3
4
5
```

# Call Stack

Es una pila que rastrea las funciones que están siendo ejecutadas en un momento dado. Cuando una función es llamada, se coloca en la parte superior de la pila; cuando la función completa su ejecución, es retirada de la pila. Este mecanismo permite a JavaScript gestionar la ejecución de funciones en secuencia, asegurando que se completan en el orden correcto y maneja adecuadamente las llamadas anidadas y recursivas. En esencia, el call stack es el corazón de la gestión de ejecución de código en JavaScript, manteniendo un registro ordenado de las llamadas a funciones para procesarlas una tras otra.





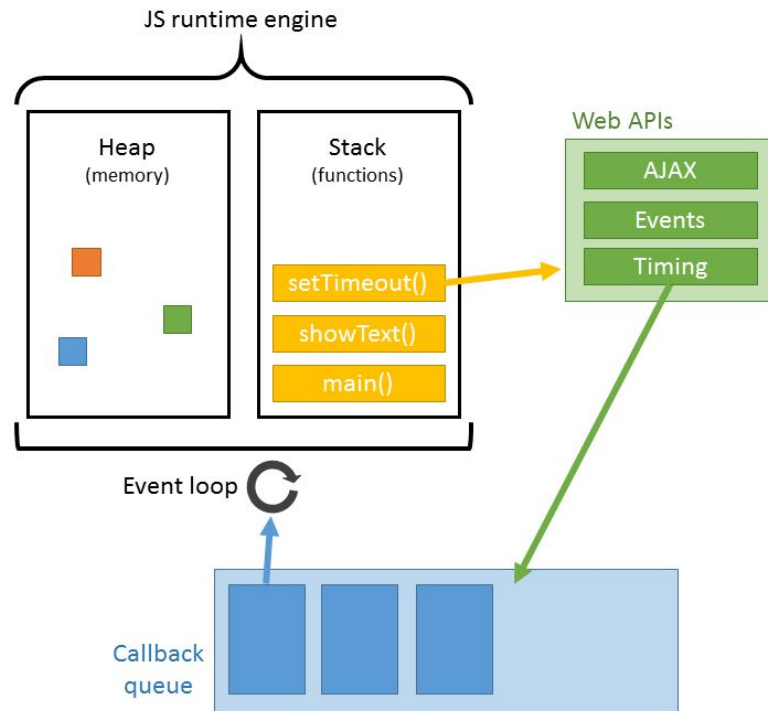
# Web API

Conjunto de interfaces de programación de aplicaciones (APIs) que proporcionan funcionalidades adicionales para interactuar con el navegador/servidor y realizar tareas que no son parte del núcleo de JavaScript.

- DOM
- Fetch API
- Promise
- `setTimeout`

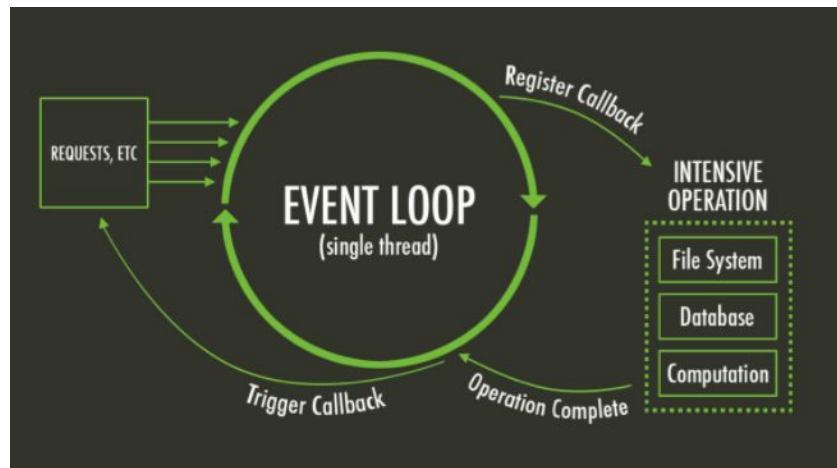
# Task Queue

Es un componente clave en la gestión de la asincronía y la concurrencia del lenguaje, especialmente dentro de su entorno de ejecución de un solo hilo como en el navegador o Node.js. Esta cola de tareas trabaja junto con el Event Loop (bucle de eventos) para orquestar el orden y la ejecución de las operaciones asincrónicas, tales como los callbacks de eventos, las promesas resueltas, y las operaciones de entrada/salida (I/O).



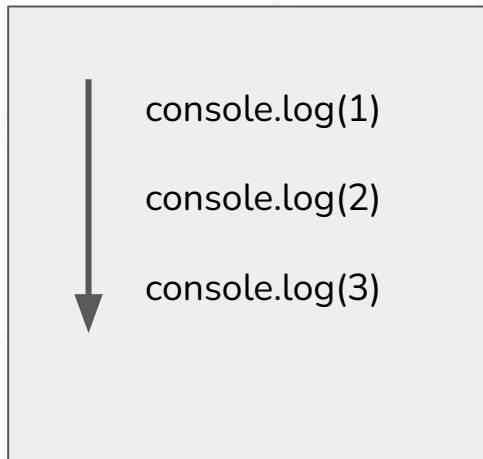
# Event Loop

Es un mecanismo fundamental en el modelo de concurrencia de JavaScript, permitiendo que el lenguaje, que es de un solo hilo, ejecute código, recolecte eventos y ejecute tareas de manera asíncrona. Es la pieza central que permite a JavaScript realizar operaciones no bloqueantes, a pesar de que puede procesar una sola instrucción a la vez en su hilo principal.

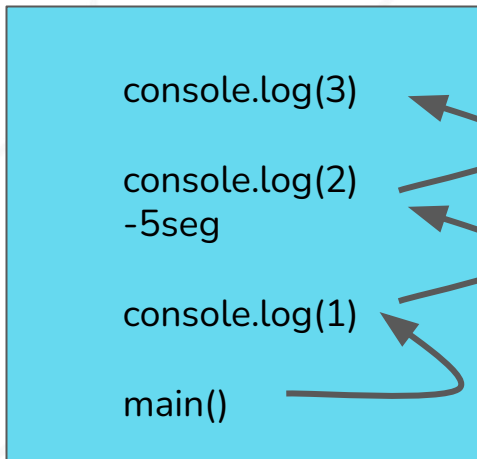


# ¿Como funciona? - Flujo normal síncrono

main.js



Pila de Ejecución  
(call stack)



Cola de Ejecución  
(callback queue)



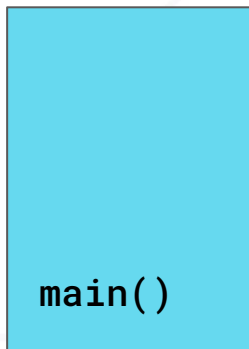
Terminal



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

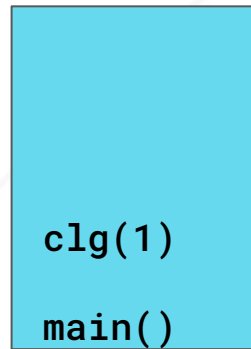
Pila de  
Ejecución  
(call stack)



Terminal



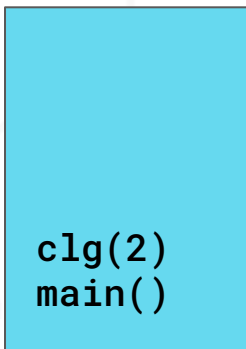
Pila de  
Ejecución  
(call stack)



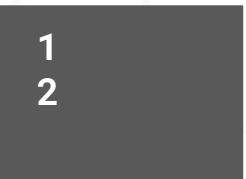
Terminal



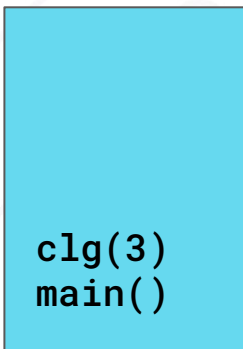
Pila de  
Ejecución  
(call stack)



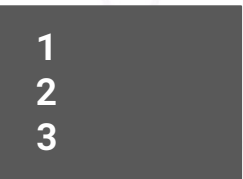
Terminal



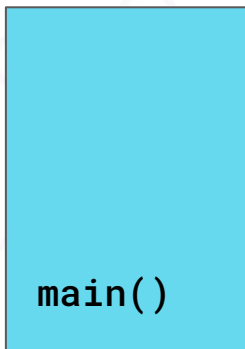
Pila de  
Ejecución  
(call stack)



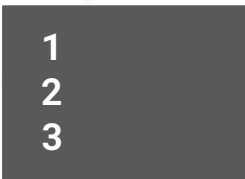
Terminal



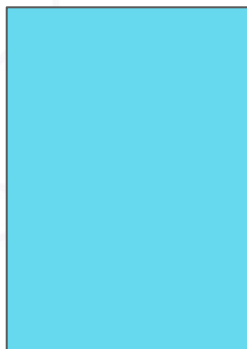
Pila de  
Ejecución  
(call stack)



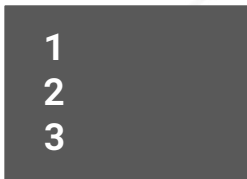
Terminal



Pila de  
Ejecución  
(call stack)



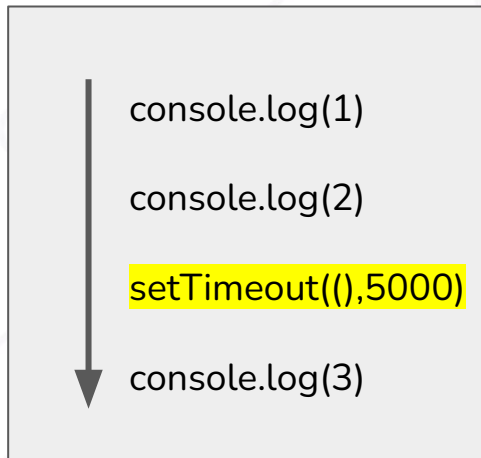
Terminal





# ¿Como funciona? - Flujo asíncrono

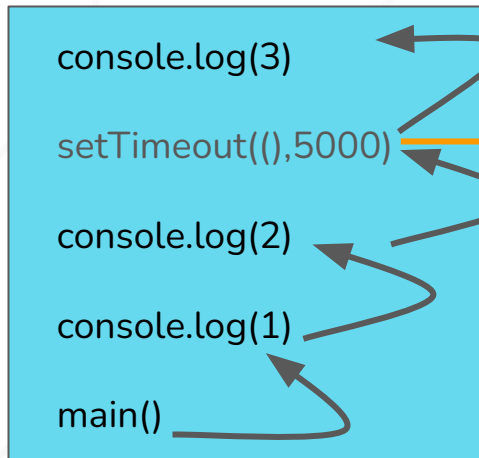
main.js



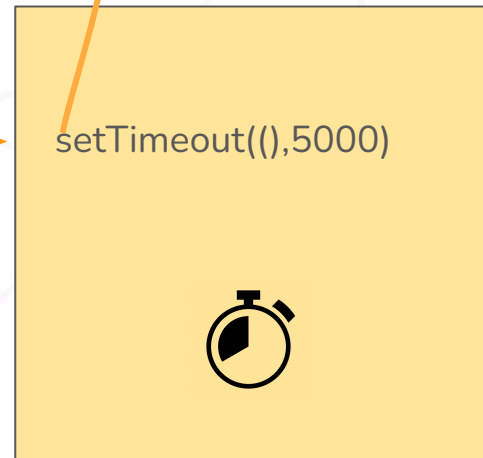
Terminal

```
1
2
3
Resultado setTimeout
```

Pila de Ejecución  
(call stack)



Cola de Ejecución  
(callback queue)



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.***