BOOTCAMP

Generación T

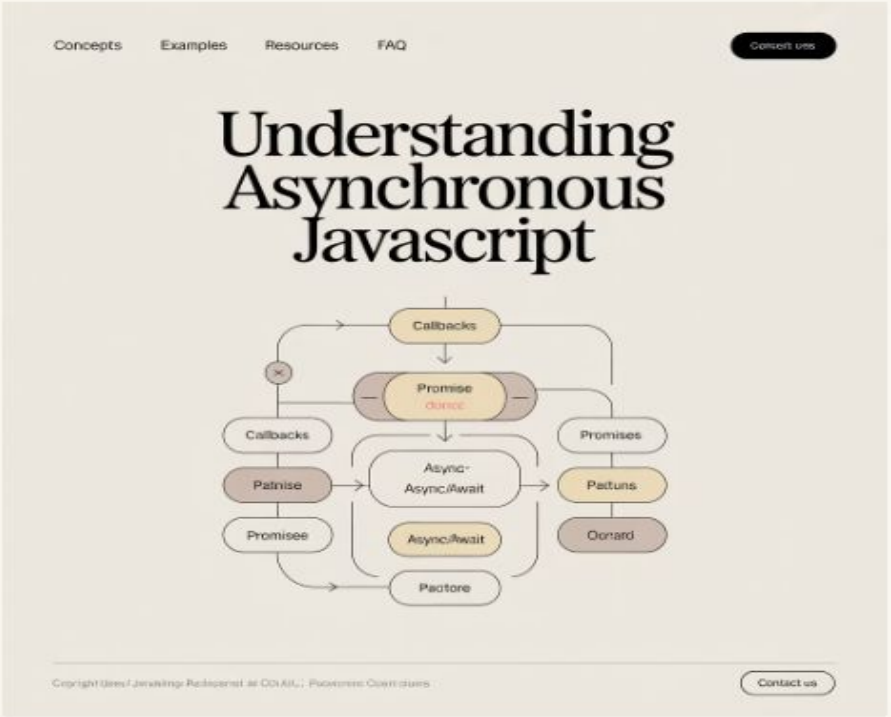
streambe

Asincronía y promesas

¿Qué es la asincronía en JavaScript?

La asincronía en JavaScript permite la ejecución de tareas que no ocurren simultáneamente con el flujo principal del programa. Esto es fundamental para:

- Obtener datos de servidores sin congelar la interfaz
- Ejecutar operaciones costosas en segundo plano
- Interactuar con APIs del navegador (geolocalización, etc.)



Modelos de programación sincrónica y asincrónica

Modelos de programación sincrónica y asincrónica

Modelo Sincrónico

- Ejecución **secuencial** de tareas
- Bloquea el hilo principal durante operaciones
- Interfaz de usuario congelada durante procesos largos

```
// Modelo sincrónico
function obtenerDatos() {
  // Bloquea hasta completar
  const datos = solicitarDatos();
  console.log(datos);
}
```

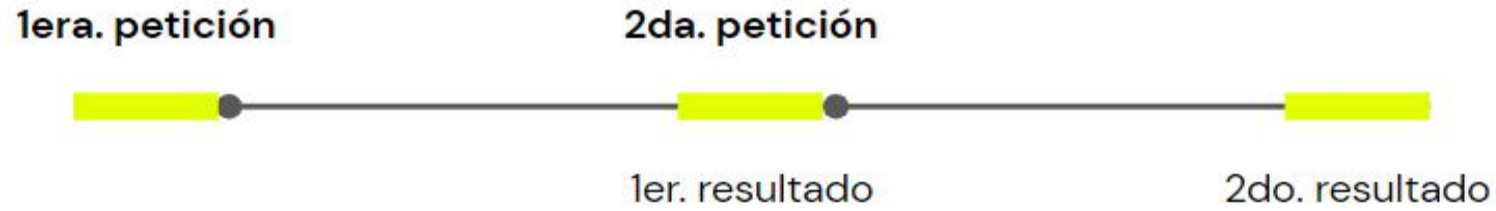
Modelo Asincrónico

- Ejecución **no bloqueante** de operaciones
- Continúa ejecutando código mientras espera resultados
- Ideal para operaciones de E/S, red y temporizadores

```
// Modelo asincrónico con promesas
function obtenerDatos() {
  solicitarDatos()
    .then(datos => console.log(datos))
    .catch(error => console.error(error));
  console.log("Continúa la ejecución");
}
```

Programación sincrónica

En este modelo, nuestro programa funciona de manera **lineal**, ejecutando una acción y después otra. Sólo podemos realizar una tarea a la vez y cada tarea es bloqueante de la siguiente 🎯.



Programación asincrónica

Este modelo permite que **múltiples tareas** sucedan a la vez. Al comenzar una acción, nuestro programa sigue en ejecución; y cuando la acción termina nuestro programa es informado y consigue acceso al resultado correspondiente .

Para recordar

👍 Una de las principales ventajas del modelo asincrónico: facilita el manejo de programas que realizan múltiples acciones a la vez.
✋ Uno de sus principales riesgos: puede dificultar la comprensión de aquellos programas que tienden a seguir una única línea de acción

¿Cómo funciona la asincronía en este contexto?
¿Cómo resolver situaciones comunes?

Problemas del modelo síncrono

1 Bloqueo de la interfaz

El hilo principal queda bloqueado mientras se esperan respuestas externas, provocando que la aplicación parezca congelada para el usuario.

2 Experiencia degradada

Las aplicaciones interactivas sufren una pobre experiencia de usuario cuando no responden de forma inmediata a las interacciones.

3 Mantenimiento complejo

El código secuencial se vuelve difícil de escalar y mantener conforme la aplicación crece en complejidad y funcionalidades.

Call Stack

El Call Stack en JavaScript

El **Call Stack** o pila de llamadas es una estructura LIFO (Last In, First Out) que:

- Registra la posición de ejecución actual del programa
- Almacena temporalmente las llamadas a funciones
- Gestiona el orden de ejecución del código sincrónico

```
function saludar() {
  console.log("Hola");
}

function iniciar() {
  saludar();
  console.log("Programa iniciado");
}

iniciar();
```

En código sincrónico, el Call Stack procesa una operación a la vez.

Llamada a función

Se añade al tope de la pila

Ejecución

Se ejecuta la función actual

Finalización

Se elimina de la pila al completarse

CALL STACK

¿Cómo es el proceso de **Call Stack**? 📞

Cuando se está a punto de ejecutar una función, ésta es añadida al stack. Si la función llama a la vez, a otra función, ésta es agregada sobre la anterior:

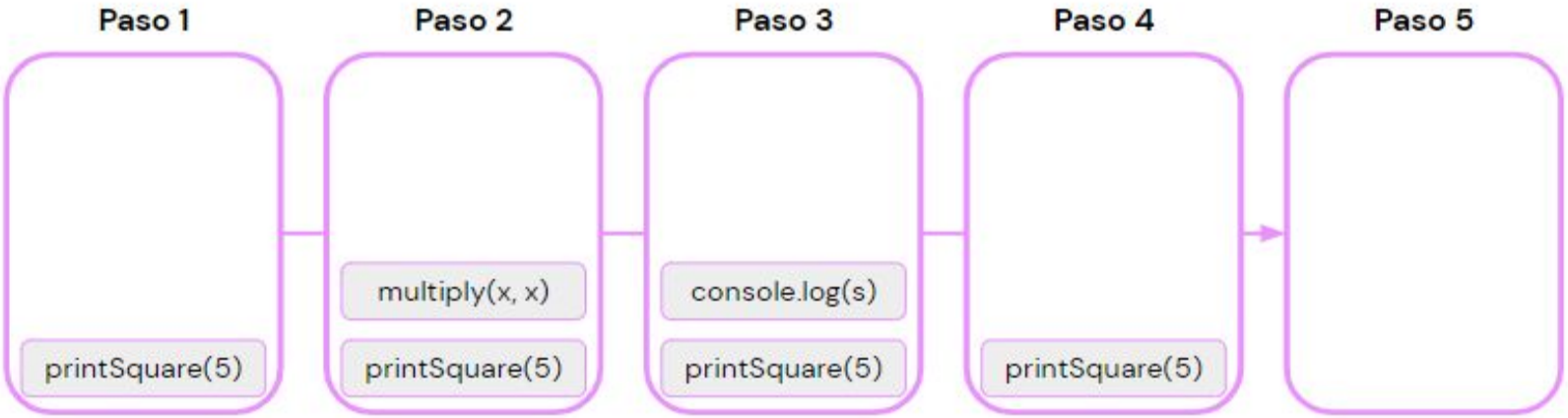
```
function multiply (x, y) {
    return x * y;
}

function printSquare (x) {
    let s = multiply(x,
x);
    console.log(s);
}

printSquare(5);
```

CALL STACK

Los estados de Call Stack son:



Es una **lista de tareas** de JS a ejecutar durante el programa 📝.
Cada nueva instrucción se agrega en el orden que corresponde al stack y el motor de JS resuelve una a una.

Event Loop

Event Loop en JavaScript

El **Event Loop** es el mecanismo que permite a JavaScript manejar operaciones asíncronas con un solo hilo de ejecución.

Call Stack

Gestiona la ejecución del código sincrónico siguiendo la estructura LIFO.

Callback Queue

Almacena las funciones callback listas para ejecutarse cuando el Call Stack esté vacío.

Event Loop

Monitorea continuamente el Call Stack y la Queue, moviendo callbacks al Stack cuando está disponible.

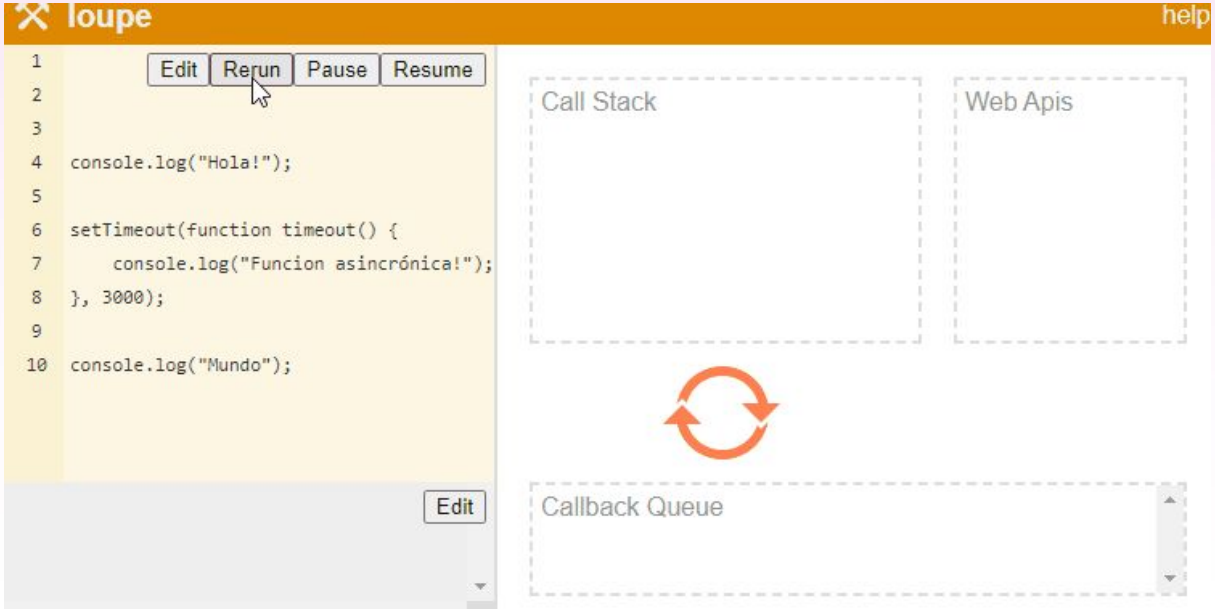
```
setTimeout(() => {
  console.log("Esto se ejecuta después");
}, 0);

console.log("Esto se ejecuta primero");
```

Aunque el timeout sea 0ms, la función callback se procesa después debido al Event Loop.

Loupe

Loupe es una visualización que nos ayuda a comprender cómo interactúan entre sí call stak, event loop y callback queue. Podemos escribir código sincrónico y asincrónico y ver cómo es el funcionamiento de estas piezas en conjunto 🧩:



ClearInterval & clearTimeout

ClearInterval & clearTimeout

En caso de querer remover un Intervalo, utilizamos la función **clearInterval ()**. También podemos detener la ejecución de un setTimeout invocando **clearTimeout ()**.

ClearInterval & clearTimeout

Cuando llamamos un setInterval() éste retorna una referencia al intervalo generado, el cual podemos almacenar en una variable. Es esta referencia la que debemos pasar a la función clearInterval para que la limpieza tenga efecto:

```
let counter = 0
const interval = setInterval(() => {
    counter++
    console.log("Counter: ", counter)

    if (counter >= 5) {
        clearInterval(interval)
        console.log("Se removió el intervalo")
    }
}, 1000)
```

```
Counter:  1
Counter:  2
Counter:  3
Counter:  4
>
```

ClearInterval & clearTimeout

Funciona igual con los timeout. Si guardamos en una variable la referencia al timeout generado, podemos usarla para removerlo luego. En el siguiente caso, el timeout generado nunca llega a ejecutarse:

```
console.log("Inicio")

const fin = setTimeout(() => {
    console.log("fin")
}, 2000)

clearTimeout(fin)
```

¿Alguna duda?

streambe GENERACIÓN T

Muchas gracias.

streambe GENERACIÓN T

Nuestras Redes

www.generaciont.org
www.streambe.com
www.instagram.com/generaciont_ar
www.tiktok.com/@generaciont
generaciont@generaciont.org
Cel: 11 61331747

GENERACIÓN T

streambe