

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.
5. Usar Callbacks.

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

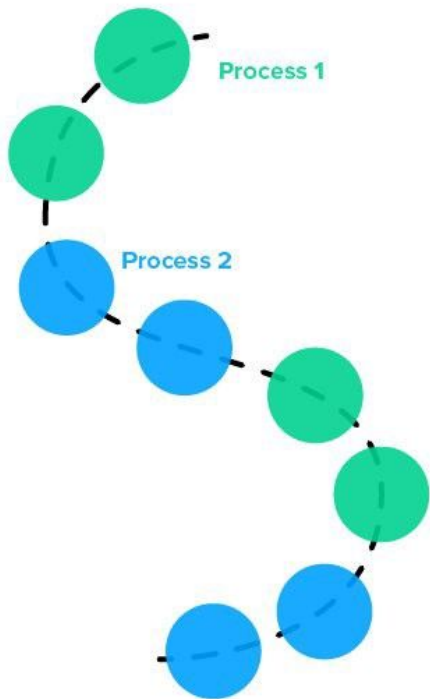
¿Que es paralelismo?

Es la ejecución simultánea de múltiples tareas

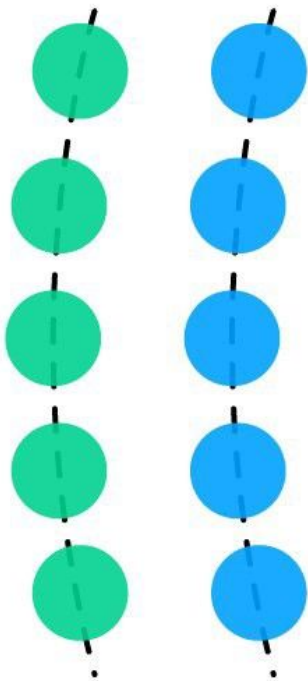
¿Que es concurrencia?

Es la capacidad de un sistema para manejar múltiples tareas de manera aparentemente simultánea

Concurrency



Parallelism



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).

ASINCRONISMO EN JS

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.

Entonces... En el asincronismo:

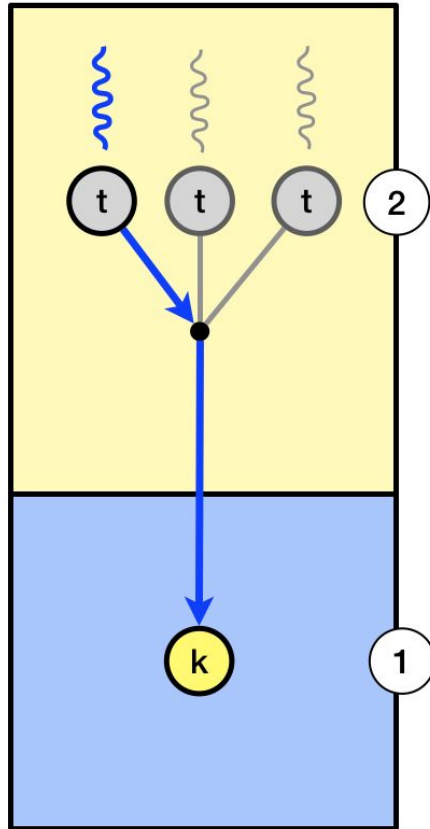
Las tareas pueden ser realizadas más tarde, lo que hace posible que una respuesta sea procesada en diferido.

Hilos de Ejecución

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Hilos de ejecución

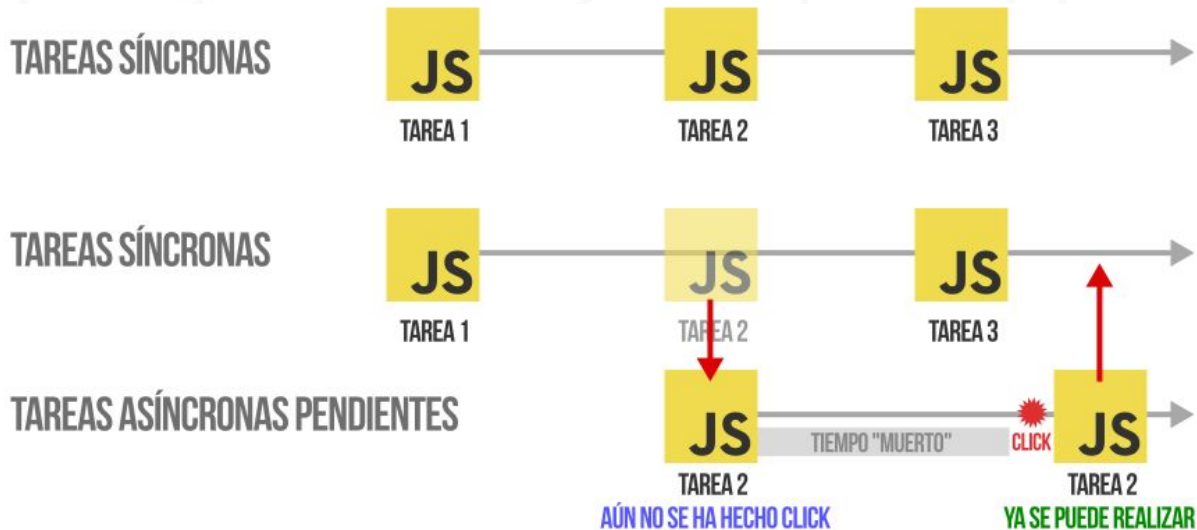


JAVASCRIPT ES:

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



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 procesamiento de la respuesta se hará en algún momento futuro.

FORMAS DE MANEJAR EL ASINCRONISMO

Callbacks, Promises, Async/await

DEV.FX
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.