



**¡Les damos la
bienvenida!**

¿Comenzamos?

Esta clase va a ser

- grabada

Clase 03. DESARROLLO AVANZADO DE BACKEND

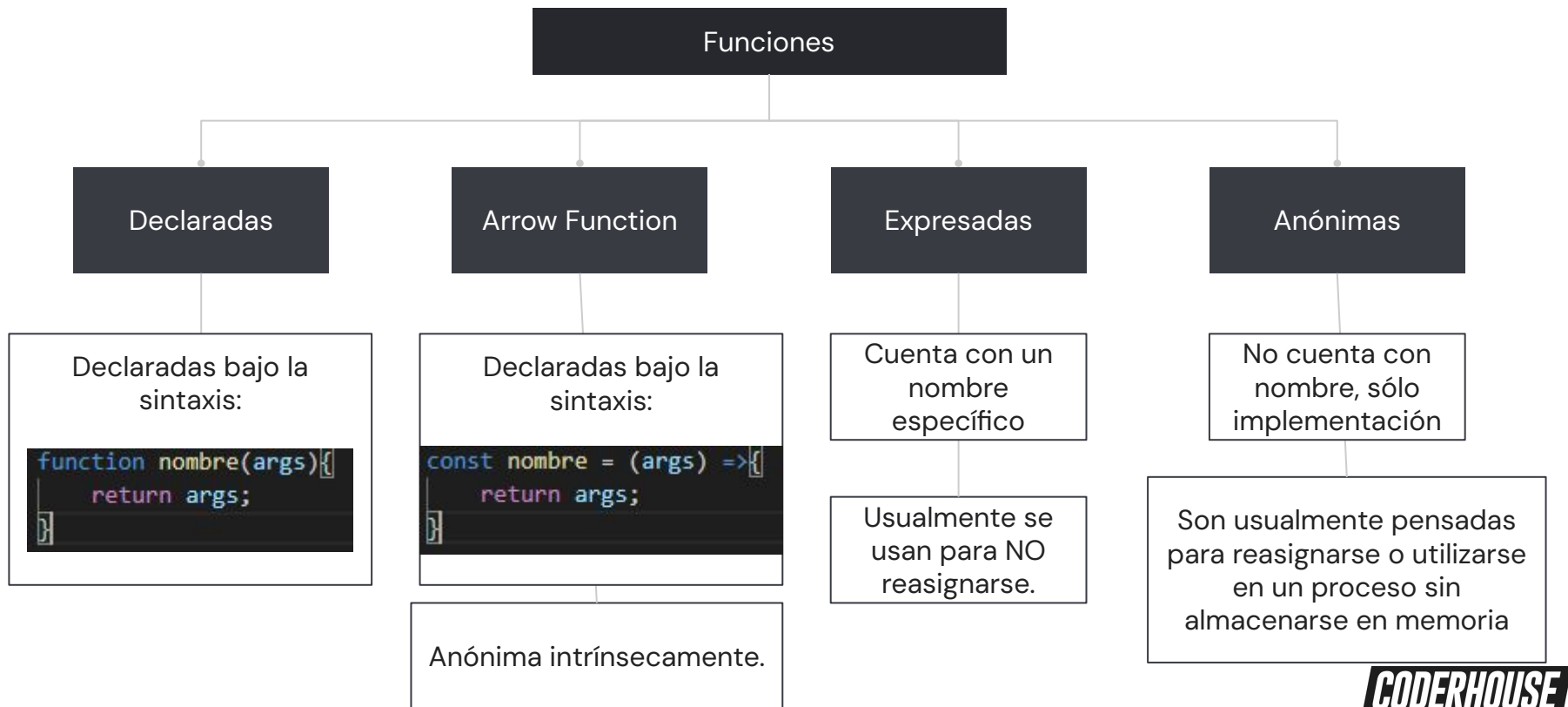
Programación sincrónica y asincrónica

Objetivos de la clase

- Repasar las funciones en Javascript
- Comprender un callback y cómo se relacionan las funciones con éstos.
- Usar promesas en Javascript
- Concretar la diferencia entre programación sincrónica y asincrónica.



MAPA DE CONCEPTOS



Funciones en Javascript



Para pensar

- ✓ ¿Por qué querríamos reasignar una función?
- ✓ ¿Por qué querríamos utilizar una función sin definirla primero?

Callbacks

Un callback es una función como cualquier otra, la diferencia está en que ésta se pasa como parámetro (argumento) para poder ser utilizado por otra función.

Permite que entonces las funciones ejecuten operaciones adicionales dentro de sí mismas

Cuando pasamos un callback, lo hacemos porque no siempre sabemos qué queremos que se ejecute en cada caso de nuestra función.

Algunos ejemplos donde has utilizado callbacks (aunque no lo creas) son:

- ✓ El método onClick en frontend
- ✓ El método forEach
- ✓ El método map o filter



Ejemplo map con callback

- ✓ Utilizaremos la función map, haciendo énfasis en el callback, para entenderlo de manera convencional
- ✓ Se explicará el funcionamiento interno de la función map para analizar el momento de utilización del callback.



Ejemplo descomposición de función map

- ✓ Se hará descomposición de la función map para poder analizarla por dentro. El objetivo es localizar en qué punto la función “map” llamaría de manera interna el callback



Ejemplo callback con operaciones


- ✓ Se crearán cuatro funciones: sumar, restar, multiplicar y dividir.
- ✓ Además, se proporcionará otra función operación, que recibirá como callback cualquiera de estas tres funciones para ejecutarla.

¡Importante!

Mientras más callbacks vamos anidando (según el tamaño del proceso) vamos formando una pirámide horizontal. en nuestro código, a esto se le conoce como **CALLBACK HELL** (también conocida como Pyramid of Doom por su forma).

Cómo reconocerlos

Si estás trabajando con callbacks y tu código comienza a tomar esta forma... ¡Mucho cuidado, hay que cambiar de estrategia!



```
pan.pourWater(function() {  
  range.bringToBoil(function() {  
    range.lowerHeat(function() {  
      pan.addRice(function() {  
        setTimeout(function() {  
          range.turnOff();  
          serve();  
        }, 15 * 60 * 1000);  
      });  
    });  
  });  
});
```

pyramid of doom



Para pensar

Si los callbacks pueden presentar este callback Hell, ¿entonces no debería utilizarlos?

¿Cuándo utilizamos los callbacks?

Promesas

Promesas

Es un objeto especial que nos permitirá **encapsular** una operación, la cual reacciona a **dos posibles** situaciones dentro de una promesa:

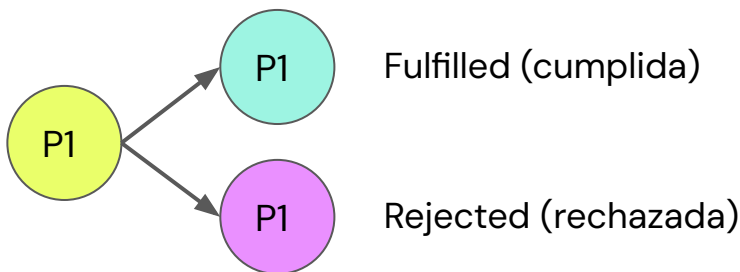
- ✓ ¿Qué debería hacer si la promesa se cumple?
- ✓ ¿Qué debería hacer si la promesa no se cumple?

The logo for JS Promises, featuring the letters 'JS' in a bold, yellow, sans-serif font inside a black square, followed by the word 'Promises' in a black, sans-serif font.

Una promesa funciona muy similar al mundo real

Al prometerse algo, es una promesa en estado pendiente (pending), no sabemos cuándo se resolverá esa promesa. Sin embargo, cuando llega el momento, se nos notifica si la promesa se cumplió (**Fulfilled**, también lo encontramos como Resolved) o tal vez, a pesar del tiempo, al final nos notifiquen que la promesa no pudo cumplirse, se rechazó (**Rejected**).

Promesas en Javascript





Ejemplo

- ✓ Se creará una promesa, haciendo énfasis en los casos de resolución (resolve) y en los casos de rechazo (reject).
- ✓ Profundizar sobre los operadores "then" y "catch" y "finally"



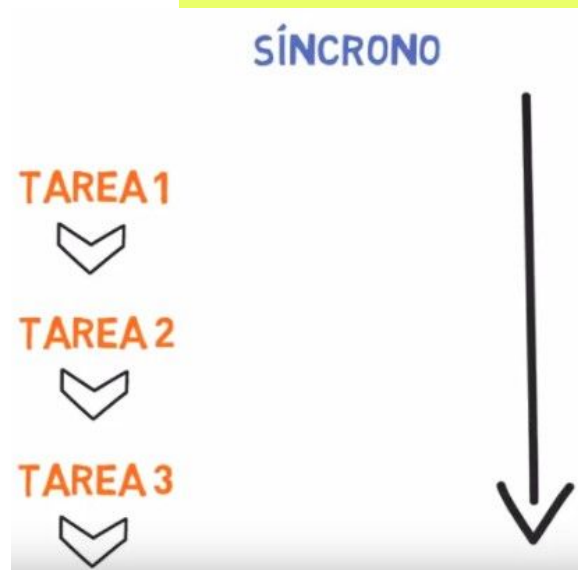
Break

¡10 minutos y volvemos!

Sincronismo vs Asincronismo

Sincronismo

Hace algunos ayeres, cuando se te enseñó a programar, entendiste que las instrucciones se ejecutaban **en cascada**, es decir, que la tarea 1 debía finalizar para que pudiera comenzar la ejecución de la tarea 2, y la tarea 2 finalizar para ejecutar la tarea 3, etc.



Importante

Las operaciones síncronas son **bloqueantes**, esto significa que las otras tareas no pueden comenzar a ejecutarse hasta que la primera no haya terminado de ejecutarse.

Asincronismo

Si lo que buscamos es que las tareas trabajen “en paralelo”, entonces debemos buscar la manera de programar instrucciones **asíncronas**, lo cual significa que cada una seguirá el hilo de resolución que considere su ritmo.

Hay que ser cautelosos al utilizarlas, ya que:

- ✓ No controlamos cuándo terminará, sólo cuándo comienza.
- ✓ Si una tarea depende del resultado de otra, habrá problemas, pues esperará su ejecución en paralelo

ASÍNCRONO

TAREA 1



TAREA 2



TAREA 3



Importante

Las operaciones asíncronas son **no bloqueantes**, esto significa que las tareas pueden irse ejecutando en paralelo y no esperar por las demás tareas.

Async / Await

Async / Await

Surge entonces en Javascript el soporte para Async – Await, unas palabras reservadas que, trabajando juntas, permiten gestionar un entorno asíncrono, resolviendo las limitantes del .then y .catch

- ✓ **async** se colocará al inicio de una función, indicando que **todo el cuerpo de esa función deberá ejecutarse de manera asíncrona**
- ✓ **await** servirá (como indica su nombre) para **esperar** por el resultado de la promesa y extraer su resultado.
- ✓ Al ser operaciones que podrían salir bien, **PERO TAMBIÉN MAL**, es importante encerrar el cuerpo en un bloque try {} catch {}

Solución





Ejemplo en vivo

- ✓ Explicación sobre el uso de una función asíncrona aplicando `async/await`.
- ✓ Usaremos la misma función de la promesa con la que hemos trabajado

Calculadora positiva con promesas

¿Cómo lo hacemos? **Se crearán un conjunto de funciones gestionadas por promesas y un entorno ASÍNCRONO donde podremos ponerlas a prueba**

- ✓ Definir función suma:
 - Debe devolver una promesa que se resuelva siempre que ninguno de los dos sumandos sea 0
 - En caso de que algún sumando sea 0, rechazar la promesa indicando "Operación innecesaria".
 - En caso de que la suma sea negativa, rechazar la promesa indicando "La calculadora sólo debe devolver valores positivos"
- ✓ Definir función resta:
 - Debe devolver una promesa que se resuelva siempre que ninguno de los dos valores sea 0
 - En caso de que el minuendo o sustraendo sea 0, rechazar la promesa indicando "Operación inválida"
 - En caso de que el valor de la resta sea menor que 0, rechazar la promesa indicando "La calculadora sólo puede devolver valores positivos"

Calculadora positiva con promesas

- ✓ Definir una función multiplicación:
 - Debe devolver una promesa que se resuelva siempre que ninguno de los dos factores sea negativo
 - Si el producto es negativo, rechazar la oferta indicando "La calculadora sólo puede devolver valores positivos"
- ✓ Definir la misma función división utilizada en esta clase.
- ✓ Definir una función asíncrona "cálculos", y realizar pruebas utilizando `async/await` y `try/catch`

¿Preguntas?

Muchas gracias.