Antes de empezar a hablar de las promesas, de asincronia, de callbacks, debemos hablar del event loop, quien permite esta magia, que no es magia, es programacion, que hemos hecho en algun momento.

Al principio, puede ser dificil de comprender, para ello haré un esfuerzo de explicarlo de la mejor manera.

Por qué debería importarte esto? JavaScript, es de un solo hilo, cómo así ? que solo atiende o ejecuta o realiza una tarea a la vez, por lo general no puede ser la gran cosa, pero imaginate que estas ejecutando una tarea y esta lleva más de 30 segundos, JavaScropt se ejecuta en el subproceso principal del navegador de forma predeterminada, por lo tanto todas las interfaces de usuario estaran atascadas, es 2023, obviamente lo que todos queremos es VELOCIDAD.

Por suerte, el navegador nos brinda algunas caracteristicas que el motor de JS no lo hace: una API web. Esto inclye la API DOM, seTimeout, Solicitudes http, que pueden ayudarnos a crear como un comportamiendo asincrono y sin bloqueo.  
  
Cuando nosotros invocamos una función, se agrega a algo llamado la pila de llamdas. La pila de llamadas hace parte del motor de JS, esto no es del navegador. Es una pila, por ende es una LIFO el ultimo en entrar, es el primero en salir o el primero en entrar es el ultimo en salir.

function greet () {

return 'Hola'

}

function response() {

return setTimeout (() => {

return 'hola!!!'

},1000)

}

greet()

response()

como vemos el response devuelve esto como tal, pero que pasa al ejecutar response, esta explicitamente no duevuelve un valor y el setTimeout por naturaleza no devuelve nada, pero recodando que es una pila, por qué salio greet de primero?...

recordemos que setTimeout, no los proporciona la web api, permitiendo simular un comportamiento asincrono, en este caso de 1 segundo.

cuando se llama a response, se ejecuta el setTimeout en 1000ms, esa funcion no se devuelve directamente a la pila de llamadas, sino que se pasa a una cola, para que espere su turno, (explico lo de la web api, cola, callstack)

el event loop es quien contencta la cola con el call stack.

Hago el ejemplo

const foo = () => console.log(“primero”)

const bar = () => setTimeout(() => console.log(“second), 500)

const baz = () => console.log(“tercero”)  
  
bar();  
foo();  
baz();