**Introducción al Asincronismo.**

JavaScript es un lenguaje de programación asíncrono y no bloqueante con un manejador de eventos llamado **Event Loop** implementado en un único hilo para sus interfaces de entrada y salida.

¿Qué es asincronismo?

En simples palabras es una acción que no ocurre al mismo tiempo.

El **Memory Heap** sirve para almacenar las asignaciones de variables y funciones y estas no se almacenan de manera ordenada y hay una interacción con el **Callstack**.

El **Callstack** es la pila de tareas que ejecuta JavaScript de manera síncrona, o sea una por una y cuando una tarea que no le corresponde como una API, este le pasa la tarea al navegador, el navegador realiza la tarea y se la pasa al **Callback Queue**.

Y aquí el trabajo del **Event Loop** es de estar observando que la pila de tareas este vacía y que el **Callback Queue** tenga la tarea terminada para ser enviada a la lista de tareas (**Callstack**) para poder ser ejecutada.

En pocas palabras el **Event Loop** es un intermediario/observador entre la pila y cola para la distribución de tareas unidireccional.

**Promesas.**

Una promesa originalmente esta pendiente. Cuando llegamos a **resolve** es cuando la promesa pasa a estar resuelta, si llamamos a **reject** pasa a estar rechazada. Usualmente cuando es rechaza obtenemos un error que nos va a indicar la razón del rechazo. Cuando una promesa se resuelve se ejecuta la función que pasamos al método .**then**, si la promesa es rechazada se ejecuta la función **.catch**, de esta forma podemos controlar el flujo de datos.

Cuando necesitamos hacer muchos llamados en los que podemos caer en callback hell, el cual es un código complejo y difícil de mantener. Una solución a este problema son las **Promesas**.

La promesa es un objeto que representa una terminación o un fracaso de una operación asíncrona. Esencialmente, una promesa es un objeto devuelto al cual se adjuntan funciones callback, en lugar de pasar callbacks a una función.

**Callbacks vs Promesas vs Async/Await**

**Callback**

Ventajas

* Cuando una función recibe otra función significa que va a ejecutarse sin ningún problema.
* Son universales, corren en cualquier navegador. Nos da garantía que se va a correr en cualquier lado.

Desventajas

* La composición es tosca, ya que se anida una función dentro de otra.
* Llega a ser compleja y se llega a un Callback Hell lo cual no podría ser soportada para el desarrollador ya que no se puede entender.
* Flujo poco intuitivo.

**Promesas**

Ventajas

* Fácilmente enlazables con los then y return.
* Es más intuitivo.
* Es más fácil de leer.
* Es poderoso, nos permite tener una gran capacidad de trabajar con asincronismo.

Desventaja

* NO maneja excepciones.
* Podemos ser propensos a errores si no retornamos el siguiente llamado.
* Requiere un polyfill para funcionar en todos los navegadores se tiene que transpilar el código como una herramienta para que pueda correr en todos los navegadores.

**Async Await**

Ventajas

* Se usa el try/catch.
* Se pueden manejar las excepciones y poder trabajar de manera fluida en la construcción del programa.
* Se pueden leer mejor.
* Da facilidad para controlar el flujo de la aplicación.

Desventajas

* Se tiene que esperar por cada uno de los llamados.
* Si se quieren hacer otros llamados hay que esperar que los await se ejecuten.
* Requiere Polyfill como las promesas.