Tarea 9: Desarrollo WEB

CallBacks

Piramide DOOM Eternal

```
let flagHola = false;
let flagMundo = false;

function callBackHellHola(){
  setTimeout(function primerHola(){
    console.log("Hola1");
    setTimeout(function segundoHola(){
      console.log("Hola2");
      setTimeout(function tercerHola(){
        console.log("Hola3");
        setTimeout(function cuartoHola(){
          console.log("Hola4");
          setTimeout(function quintoHola(){
            console.log("Hola5");
            flagHola = true;
          }, 1000);
        }, 1000);
      }, 1000);
    }, 1000);
  }, 1000);
}

function callBackHellMundo(){
  setTimeout(function primerMundo(){
    console.log("Mundo1");
    setTimeout(function segundoMundo(){
      console.log("Mundo2");
      setTimeout(function tercerMundo(){
        console.log("Mundo3");
        setTimeout(function cuartoMundo(){
```

```
            console.log("Mundo4");
            setTimeout(function quintoMundo(){
              console.log("Mundo5");
              flagMundo = true;
            }, 2000);
          }, 2000);
        }, 2000);
      }, 2000);
    }, 2000);
}

function ifEnd(){
  if(flagHola && flagMundo){
    console.log("FIN");
  } else{
    setTimeout(ifEnd, 100);
  }
}

callBackHellHola();
callBackHellMundo();
setTimeout(ifEnd, 5000);
```

Primero declaramos 2 variables booleanas, cuyo valor inicial es falso para ambos, esto tendrá más sentido en la línea 43.

Las funciones recrean lo visto en clase de una pirámide de DOOM, en esta primera función llamada callBackHellHola() se empiezan con un setTimeout cuyo único parámetro es una función que solo hace un console.log en ese se imprime el hola. Esta serie continua otras 4 veces más hasta llegar a la línea 15 en donde la primer variable booleana es puesta como true. De ahí la función cierra con el tiempo en milisegundos, específicamente 1 segundo de espera, para que empiece a imprimir la cadena de callBacks.

La segunda función llamada callBackHellMundo() recrea lo mismo pero con los cambios de que el console.log imprime Mundo en lugar de Hola además que los setTimeout's tienen 2 segundos de espera en la cadena de console.log's.

Finalmente la función ifEnd(), se pregunta si las dos variables booleanas son verdaderas, que lo son, imprimen por medio del console.log el mensaje 2 “FIN”. En las llamadas a las funciones utilizamos callBackHellHola(), callBackHellMundo() y la función setTimeout() con los parámetros ifEnd y el tiempo de 5 segundos

Promises

Pinky Promise

```
function printHola(value){
   return new Promise(function (resolve,reject){
      setTimeout(() => {
         console.log("Hola " + value);
         resolve(value + 1);
      }, 1000)
   });
}

let pH = printHola(1)
   .then(result => printHola(result))
   .then(result => printHola(result))
   .then(result => printHola(result))
   .then(result => printHola(result))
   .then(() => Promise.resolve());

function printMundo(value){
   return new Promise(function (resolve, reject){
      setTimeout(() => {
         console.log("Mundo " + value);
         resolve(value + 1);
      }, 2000)
   });
}

let pM = printMundo(1)
```

```
  .then(result => printMundo(result))
  .then(result => printMundo(result))
  .then(result => printMundo(result))
  .then(result => printMundo(result))
  .then(() => Promise.resolve());

pH.then(() => pM)
  .then(() => console.log("FIN"));
```

La función printHola(value) recibe un valor, esta función retorna una promesa la cual retorna una función con los parámetros resolve y reject, aunque solo usaremos resolve.

La función de la promesa empieza con un setTimeout de 1 segundo la cual imprime en consola un Hola el cual está concatenado con el value que recibe al comienzo de printHola. Para evitar que el valor no cambie, el resolve de la promesa incrementa el value en uno.

De la línea 10 a 15 creamos una variable pH que es una promesa, claramente printHola(1) y uno es el value que se va a ir incrementando, después de eso .then, se pasa el result para que sea el argumento de la función. Para este entonces 1 ya cambió a 2.

En la línea 15 ya tendría el valor 6 por lo que crea una nueva promesa que solo se resuelve para cerrar el ciclo.

Para el caso de la función printMundo(value) es más de lo mismo con el mayor cambió es que espera 2 segundos y lo mismo con la variable pM en donde crean más promesas cuyo resultado sea el anterior + 1.

En la línea 33 y 34 la promesa pH se ejecuta y en el momento en el que se ejecute este desencadenará en que pM también inicialice para que en cuanto ambas funciones términen, se imprime en consola el mensaje “FIN”.

Async/Await


```
function delay(ms) {
  return new Promise(function (resolve) {
    setTimeout(() => resolve(), ms);
  });
}
async function printHola(){
  for(let i = 1; i < 6; i++){
    let response = await delay(1000);
    console.log("Hola" + i);
  }
  return Promise.resolve();
}
async function printMundo(){
  for(let i= 1; i < 6; i++){
    let response = await delay(2000);
    console.log("Mundo" + i);
  }
  return Promise.resolve();
}

Promise.all(
  [
  printHola(),
  printMundo()
  ]).then(()=> console.log("fin"));

(async function tareaFin(){
  await Promise.all([printHola(), printMundo()] );
  console.log("FIN");
})();
```

El Bloque de código que va desde la línea 1 a la 5 es una función de delay que como su nombre da a entender ayuda a asignar un tiempo de espera. El funcionamiento de esta función recibe un tiempo en milisegundos, este retorna una promesa que recibe un resolve en este resolve es que se procesa el tiempo que se atrasará

El bloque de código de las líneas 7 a 13 es una función printHola que imprime la palabra "Hola" seguida de números del 1 al 5. El funcionamiento de esta función es asíncrona y utiliza un bucle para repetir la acción 5 veces con un intervalo de 1 segundo entre cada impresión. Este bloque es casi parecido a la función printMundo() con la única diferencia del tiempo de espera es 2 segundos.

La instrucción Promise.all. El funcionamiento de este método recibe un array de promesas y espera a que todas se completen para luego ejecutar una acción final.

La última función es anónima espera a que todas las funciones términen para que imprima un el mensaje de “FIn” en la consola