EJ3. Programación asíncrona en JS y AJAX

Ejercicios para practicar: Desarrollar los problemas propuestos

Lectura de código: Promesas

Para los siguientes fragmentos de código

- ¿Cuál es la salida de cada uno de ellos?
- ¿Explica por qué la salida es esa y no otra?
- ¿Alguna parte del código se ejecuta de forma asíncrona?

```
// Fragmento 1

console.log("Empezamos...");

const promise1 = new Promise((resolve, reject) => {
  console.log("Orden 1");
})

console.log("Se acabó!!");
```

```
// Fragmento 2

console.log("Empezamos...");

const promise1 = new Promise((resolve, reject) => {
  console.log("Orden uno")
  resolve("Orden dos")
})

promise1.then(respuesta => {
  console.log(respuesta)
})

console.log("Se acabó!!");
```

```
// Fragmento 3

console.log("Empezamos...");

const promise1 = new Promise((resolve, reject) => {
  console.log("Orden uno")
```

```
  resolve("Orden dos")
  console.log("Orden tres")
})

promise1.then(respuesta => {
  console.log(respuesta)
})

console.log("Se acabó!!");
```

```
// Fragmento 4

console.log("Empezamos...");

const promise1 = new Promise((resolve, reject) => {
  console.log("Orden uno")
})

promise1.then(respuesta => {
  console.log("Orden dos")
})

console.log("Se acabó!!");
```

```
// Fragmento 5

console.log("Empezamos...")

const fn = () => (new Promise((resolve, reject) => {
  console.log("Orden uno");
  resolve("Ok")
}))

console.log("Orden dos")

fn().then(res => {
  console.log(res)
})

console.log("Se acabó!!")
```

```
// Fragmento 6

console.log("Comenzamos...")

Promise.resolve("Orden uno").then((respuesta) => {
  console.log(respuesta)
})

Promise.resolve("Orden dos").then((respuesta) => {
  console.log(respuesta)
})

console.log("Se acabó!!")
```

Promesa aleatoria

- Plantear una promesa que dependa del valor que arroje una función que genera números entre cero y diez. Cuando el número sea menor o igual 5 la promesa debe ser resuelta pasados 3 segundos, caso contrario debe ser rechazada igualmente transcurridos tres segundos. Independientemente del resultado de la promesa mostrar el mensaje "Promesa acabada" para indicar que el proceso terminó
- Escribe el código correspondiente. Puedes presentar el resultado por consola o con la función alert(). Por supuesto también puedes hacerlo manipulando el DOM y presentando los resultado en un documento HTML

Claves:

- Recuerda el código genérico para crear una promesa en JS y el correspondiente para consumirla
- Recuerda la diferencia entre **código ejecutor** y **código consumidor**

```
// Definición de la promesa
const promise = new Promise ((resolve, reject) => {
    // Código ejecutor
    // comprobar código y declarar las funciones resolve y reject
});

// Consumo de la promesa
promise
    .then((mensaje) => {
        // codigo consumidor
    })
    .catch((error) => {
        // codigo que maneja el error
    })
    .finally(() => {
        // codigo que se ejecuta en cualquier caso
    })
```

Promisificación

Se denomina *promisificación* a la conversión de código asíncrono basado en callbacks a código basado en promesas. No siempre es posible hacerlo por unas cuestiones u otras, pero cuando es posible hacer la promisificación del código se mejora la legibilidad y el mantenimiento del mismo. Vamos a trabajar esta cuestión.

Supón que tienes el siguiente código

```
function getData(callback) {
    if (Math.random() > 0.2) {
        setTimeout(() => {
            callback("Ok. Los datos han llegado");
        }, 2000);
    } else {
        callback("Error en la comunicación!!");
    }
}

getData(data => {
    console.log(data);
});
```

1. ¿Podrías explicar qué es lo que hace ese código?
2. ¿Podrías identificar en ese código las parte asíncrona y la parte síncrona?
3. Reescribe el código usando un objeto Promise qué sustituya al uso de la función callback presente en él

Uso de Async/Await

A partir de 2017 se introduce en la especificación de JS una nueva estructura para manejar código asíncrono. Esta estructura se denomina **Async/await**. Con Async/await se facilita el manejo de promesas y se consigue un código que se asemeja a una forma “más secuencial”.

Intenta reescribir el código basado en promesas obtenido en el ejercicio anterior usando la estructura Async/await

Clave: Se trata de “envolver” la promesa con una función declarada como `async` y después ejecutar el código consumidor y llamar a la función “asíncrona” con la palabra `await`

Mostrar un usuario al azar

La página https://randomuser.me/ permite obtener datos aleatorios de personas para uso en desarrollo de aplicaciones. Las instrucciones para usar su API están en su web. Míralas para ver como usarlas en el código que vas a desarrollar a continuación. La idea es hacer peticiones vía GET a la URL https://randomuser.me/api con determinados parámetros que nos interesen

Desarrolla una web donde se presente un usuario al azar con foto y datos personales.

Claves:

- En la web de randomuser.me puedes hacer pruebas de peticiones GET
- Usa JSON como formato de intercambio
- Usa la API Fetch. Es más sencilla de manejar que el objeto XMLHttpRequest
- Usa plantillas de texto para pasar los datos al documento HTML

Usuario al azar cambiante

Vamos a ampliar la funcionalidad de la web anterior añadiendo un botón que al pulsarlo cambie el usuario cargado a otro distinto

Mapa del tiempo

Vamos a trabajar con la API de AEMET de su portal de datos abiertos. Para ello hay que darse de alta como desarrollador en la web https://opendata.aemet.es/centrodedescargas/altaUsuario

Puedes acceder a AEMET OpenData para obtener información interesante para desarrollo de aplicaciones web que usan los datos de AEMET

Con la información anterior se trata de que desarrolles una página web que cargue la imagen del *mapa del día*

Claves:

- Lo primero es conseguir una API Key para poder hacer peticiones
- Usa la API Fetch en con Async/await. Es más sencillo
- Lo mejor es insertar la llamada a la API ha de insertarse en un evento de carga (***onload***) del documento HTML

Mejora:

- Introduce la posibilidad de añadir la API Key desde la propia página web

Cambio de moneda

Vamos a crear una web sencilla que nos permita hacer un cambio de divisa. Para ello usaremos la API de https://exchangeratesapi.io/. Hay que registrarse en la versión gratuita que nos permite hasta mil consultas al mes sin coste.

En la documentación (https://exchangeratesapi.io/documentation/) tienes toda la información sobre los end-points e incluso ejemplos de peticiones

Desarrolla una página HTML con un formulario que permita convertir divisas entre euros, dólares USA, libras esterlinas, yenes, 'yuanes' y francos suizos

Claves:

- Diseña el formulario HTML usando un cuadro de texto para introducir la cantidad, y dos cuadros *select* para seleccionar la divisa de origen y la de destino. Usa un botón para enviar la petición y devuelve el dato en la propia web
- Maneja el evento **click** del botón del formulario para ejecutar la petición
- Usa Fetch API

(803a46a69b12da7ea849eaf5117f5a2e)

Imagen del día de la NASA

Con todo lo que has visto hasta ahora desarrolla una página web que haga lo mismo que esta:

https://apod.nasa.gov/apod/astropix.html

Claves:

- Consigue una API Key de la NASA
- Maneja la petición con la API Fetch
- Inserta la imagen en un documento HTML al cargarse