Kahoot - Tema 9

1. ¿Qué caracteriza a la programación síncrona?

- A) Funciona únicamente con callbacks
- B) Permite ejecutar varias tareas al mismo tiempo
- C) Requiere que cada instrucción espere a la anterior ✅
- D) Utiliza promesas para evitar bloqueos

Explicación: En el modelo síncrono, el hilo de ejecución es secuencial y bloqueante; una tarea no comienza hasta que la anterior ha finalizado por completo.

---

2. ¿Qué ocurre en un programa síncrono si el servidor tarda en responder?

- A) El programa queda bloqueado esperando la respuesta ✅
- B) El programa sigue ejecutándose normalmente
- C) Se ejecutan las funciones en paralelo
- D) El programa se cierra automáticamente

Explicación: Al ser un hilo único y bloqueante, cualquier operación lenta detiene toda la ejecución del código, lo que puede "congelar" la aplicación.

---

3. La programación asíncrona permite...

- A) Obligar a esperar la respuesta de un servidor
- B) Ejecutar solo una instrucción por segundo
- C) Evitar el uso de funciones
- D) Seguir ejecutando código mientras otra operación tarda en completarse ✅

Explicación: Permite delegar tareas pesadas (como peticiones de red) a segundo plano para no bloquear el flujo principal de la aplicación.

---

4. AJAX permite...

- A) Recargar por completo la página
- B) Solo trabajar con XML
- C) Actualizar contenido sin recargar la página ✅
- D) Enviar datos sin comunicación con el servidor

Explicación: La esencia de AJAX es la asincronía, permitiendo intercambiar datos con el servidor y actualizar la interfaz sin necesidad de refrescar todo el sitio web.

---

5. ¿Qué objeto se usaba inicialmente para AJAX?

- A) XMLHttpRequest ✅
- B) FormData
- C) JSONRequest
- D) AJAXRequest

Explicación: El objeto `XMLHttpRequest` fue la API original de los navegadores para realizar peticiones HTTP asíncronas antes de que existieran alternativas modernas.

---

6. ¿Qué formato de datos se usa actualmente en AJAX?

- A) XML únicamente
- B) SQL
- C) JSON ✅
- D) CSV

Explicación: Aunque la "X" de AJAX se refiere a XML, hoy en día se utiliza JSON por ser mucho más ligero y fácil de procesar de forma nativa con JavaScript.

---

7. ¿Qué API reemplaza en gran medida a XMLHttpRequest?

- A) Fetch API ✅
- B) ServerAPI
- C) PromiseAPI
- D) AjaxAPI

Explicación: La **Fetch API** es el estándar moderno; utiliza promesas, lo que permite escribir un código mucho más limpio y fácil de mantener que el antiguo XHR.

---

8. El método fetch() devuelve...

- A) Un archivo XML
- B) Una respuesta síncrona
- C) Una promesa ✅
- D) Un objeto JSON directamente

Explicación: `fetch()` siempre devuelve una promesa que, al resolverse, entrega un objeto de respuesta (`Response`). Luego se debe usar otro método (como `.json()`) para extraer los datos.

---

9. Las callbacks suelen usarse para...

- A) Declarar variables
- B) Obtener datos de un servidor o manejar eventos ✅
- C) Crear elementos HTML
- D) Cambiar estilos CSS

Explicación: Las funciones de retorno (callbacks) se pasan como argumentos para ser ejecutadas una vez que una tarea asíncrona (como un clic o una carga de datos) ha terminado.

---

10. ¿Qué método de JavaScript ejecuta código después de un tiempo?

- A) setTimeout() ✅
- B) timeout()
- C) setInterval()
- D) wait()

Explicación: `setTimeout()` establece un temporizador que ejecuta una función de callback una vez transcurrido el tiempo especificado en milisegundos.

---

11. ¿Qué es una callback?

- A) Una excepción que detiene el programa
- B) Una variable que contiene un tiempo de espera
- C) Una función pasada como argumento a otra función ✅
- D) Una promesa rechazada

Explicación: Es una función que se entrega a otra función para que sea ejecutada ("llamada de vuelta") en un momento posterior o tras un evento.

---

12. ¿Qué problema aparece al anidar muchas callbacks?

- A) Callback break
- B) Callback chain
- C) Callback hell ✅
- D) Callback freeze

Explicación: El **Callback Hell** ocurre cuando el código se vuelve ilegible y difícil de mantener debido a múltiples niveles de anidamiento de funciones asíncronas.

---

13. ¿Qué es una Promesa en JavaScript?

- A) Una función síncrona
- B) Un tipo de dato JSON
- C) Un método de setTimeout
- D) Un objeto que representa un valor futuro ✅

Explicación: Una promesa es un objeto que representa el éxito o el fracaso eventual de una operación asíncrona, permitiendo manejar valores que aún no conocemos.

---

14. ¿Cuáles son los estados de una promesa?

- A) init, resolve, cancel
- B) start, load, finish
- C) pending, fulfilled, rejected ✅
- D) begin, running, end

Explicación: Una promesa comienza en **pending** (pendiente) y termina en **fulfilled** (completada con éxito) o **rejected** (fallida).

---

15. El método .then() se ejecuta cuando...

- A) La promesa se cancela
- B) La promesa se cumple ✅
- C) La promesa se rechaza
- D) La promesa está pendiente

Explicación: El bloque `.then()` es el manejador de éxito; se dispara únicamente cuando la promesa pasa al estado de "cumplida" (*fulfilled*).

---

16. ¿Cuál de los siguientes métodos se usa para manejar errores en promesas?

- A) .done()
- B) .catch() ✅
- C) .error()
- D) .fail()

Explicación: El método `.catch()` se encarga específicamente de capturar cualquier error o rechazo que ocurra durante el ciclo de vida de la promesa.

---

17. ¿Qué hace Promise.all()?

- A) Devuelve solo la primera promesa resuelta
- B) Cancela las promesas que fallen
- C) Ejecuta promesas una por una
- D) Ejecuta promesas y se resuelve cuando todas lo hacen ✅

Explicación: Este método toma varias promesas y devuelve una sola que se resuelve solo cuando todas las promesas del grupo han terminado con éxito.

---

18. ¿Qué palabra clave convierte una función en asíncrona?

- A) async ✅
- B) wait

- C) future
- D) promise

Explicación: La palabra clave `async` delante de una función hace que esta devuelva siempre una promesa y permite usar la palabra `await` dentro de ella.

---

19. ¿Qué palabra clave se usa dentro de una función async para esperar una promesa?

- A) await ✅
- B) wait
- C) stop
- D) hold

Explicación: `await` pausa la ejecución de la función asíncrona de forma no bloqueante hasta que la promesa se resuelve, devolviendo su valor final.

---

20. La combinación async/await ofrece...

- A) Eliminación del uso de promesas
- B) Ejecución síncrona obligatoria
- C) Equivalente a callbacks pero más legible ✅
- D) Código más difícil de leer

Explicación: Es una mejora sintáctica sobre las promesas que permite escribir código asíncrono con una estructura que se lee casi como si fuera síncrona.

---

21. ¿Qué propiedad contiene el código de estado HTTP en fetch()?

- A) code
- B) status ✅
- C) http
- D) error

Explicación: El objeto `Response` devuelto por `fetch()` incluye la propiedad `.status` para indicar el resultado de la petición (ej: 200 para éxito, 404 para no encontrado).

---

22. En una petición fetch, ¿qué método se suele usar para extraer el cuerpo en formato JSON?

- A) .getBody()
- B) .parse()
- C) .json() ✅
- D) .toJson()

Explicación: El método `.json()` lee la respuesta hasta el final y devuelve una promesa que se resuelve con el resultado de parsear el texto del cuerpo como JSON.

---

23. ¿Qué sucede si una promesa en un bloque async no se maneja con await o .catch()?

- A) El programa se detiene inmediatamente
- B) Se genera una "Unhandled Promise Rejection" ✅
- C) La promesa se resuelve automáticamente
- D) JavaScript ignora la función

Explicación: Si una promesa falla y no hay nada que capture ese error, el entorno de ejecución emitirá una advertencia de rechazo no manejado, lo cual es una mala práctica.

---

24. ¿Cuál es el orden de ejecución correcto en una promesa?

- A) then → catch → finally ✅
- B) catch → then → finally
- C) finally → then → catch
- D) then → finally → catch

Explicación: Generalmente se define el flujo de éxito (`then`), el de error (`catch`) y por último el bloque que se ejecuta siempre (`finally`).

---

25. ¿Qué ventaja principal tiene Fetch sobre XMLHttpRequest?

- A) Es más antiguo y compatible
- B) Funciona de forma síncrona por defecto

- C) Utiliza promesas y tiene una sintaxis más limpia ✅
- D) No requiere conexión a internet

Explicación: `Fetch` moderniza las peticiones de red al integrarse nativamente con el sistema de Promesas de JavaScript, evitando el complejo manejo de eventos de XHR.