|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Universidad Técnica Latinoamericana**  **Facultad de Ingeniería**  **Escuela de Ingeniería en Sistemas y TI** | | | | |
| Asignatura | Herramientas avanzadas para el desarrollo de aplicaciones. | Grupo viernes PM | Ciclo  02-2025 | Modalidad de la práctica:  Presencial |
| Docente de la Catedra | **Ing. Ámilton Abraham Rodríguez** |
| Docente de la práctica | Técnico Walter Deleon |
| Ponderación | (Todas las Practicas) Evaluación 5. Ponderación 30% DE NOTA FINAL | | | |
| **Guía para la Práctica 3: Asincronía en JavaScript** | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Apellidos, Nombres | Carnet | Firma | NOTA |
| **José Nelson Menjívar Guardado** | **0701023** |  |  |

|  |  |
| --- | --- |
| Actividad | Práctica de laboratorio #3 |
| Formato de entrega | Presentará un reporte digitalizado en Microsoft Word, Arial 12, justificado, 1.5 interlineado, con su portada |
| Objetivo de la Práctica | Que los estudiantes descubran la importancia de la programación asíncrona en JavaScript mediante la implementación de callbacks, promesas y async/await |
| Fecha de entrega | 13 de septiembre de 2025 |
| Indicaciones de la Actividad | Entrega final: Elaborar un documento en Microsoft Word donde recolecte la evidencia de la ejecución de ordenada de cada uno de los bloques de código propuestos junto a la resolución de las preguntas respectivas.  Actividades a realizar:  **Parte I: Configurando el entorno**  Dirígete al sitio <https://nodejs.org> y descarga la versión LTS para Windows. Abre el instalador e instala node en el equipo de laboratorio. Asegúrate de que la casilla Add to PATH esté marcada para usar node desde la terminal.  Abre una terminal en Windows y escribe el comando node -v. Si obtienes un número de versión la instalación fue exitosa. Esto nos permitirá ejecutar código Javascript fuera del navegador.  **Parte II: Hablemos de Sincronía**  Teoría: En un programa síncrono, las instrucciones se ejecutan línea por línea, esperando a que cada una termine antes de continuar.  Práctica: Crea una carpeta y ábrela desde VS Code. Crea un archivo llamado asincronía.js  Escribe y ejecuta el siguiente código (abre una terminal en vs code y ejecuta el comando node asincronia.js):    Responde a las siguientes preguntas:   1. ¿En qué orden aparecen los mensajes? 2. ¿Qué ocurriría si una línea tarda mucho tiempo en ejecutarse? 3. ¿Qué pasaría con la experiencia del usuario si el código se queda “bloqueado”?   **Parte III: Sincronía con setTimeout**  Teoría: setTimeout permite posponer la ejecución de una función después de un tiempo determinado, sin bloquear el resto del código.  Práctica: En el mismo archivo, escribe y ejecuta el siguiente código:    Responde a las siguientes preguntas:   1. ¿Por qué aparece “Fin” antes que el mensaje de setTimeout? 2. ¿El programa se detuvo 2 segundos o siguió ejecutándose? 3. ¿Cómo ayuda esto en aplicaciones interactivas?   **Parte IV: Callbacks**  Teoría: Un callback es una función que se pasa como argumento a otra función, y se ejecuta cuando la tarea termina.  Práctica: En tu archivo de prueba, escribe y ejecuta el siguiente código:    Responde a las siguientes preguntas:   1. ¿Qué ventaja tiene usar un callback en lugar de setTimeout suelto? 2. ¿Dónde se ejecuta el callback en este código? 3. ¿Qué pasaría si necesitamos descargar tres archivos en orden? ¿Cómo luciría el código?   **Parte V: Promesas**  Teoría: Una promesa representa un valor que estará disponible ahora, más tarde o nunca. Evita la llamada “callback hell”.  Práctica: En tu archivo de “experimentos” (estamos en un laboratorio ¿verdad?) escribe y ejecuta el siguiente código:    Responde a las siguientes preguntas:   1. ¿Qué diferencias notas con el callback? 2. ¿Dónde se especifica qué hacer después de que se descarga el archivo? 3. ¿Qué ventaja tiene encadenar .then()?   **Parte VI: Async / await**  Teoría: async/await hace que el código asincrónico se lea de forma secuencial y clara, como si fuera síncrono, pero sin bloquear.  Práctica: Escribe y ejecuta el siguiente código:    Responde a las siguientes preguntas:   1. ¿Por qué parece que el código se ejecuta línea por línea aunque es asincrónico? 2. ¿Qué ventajas tiene frente a promesas con .then()? 3. ¿Cuándo preferirías usar async/await en un proyecto real? |
| Ponderación | 12.5% de la quinta nota |

***Solución a Laboratorio:***

Descarga de Node

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Instalacion

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

**Parte II: Hablemos de Sincronía**

Teoría: En un programa síncrono, las instrucciones se ejecutan línea por línea, esperando a que cada una termine antes de continuar.

Práctica: Crea una carpeta y ábrela desde VS Code. Crea un archivo llamado asincronía.js

Escribe y ejecuta el siguiente código (abre una terminal en vs code y ejecuta el comando node asincronia.js):

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

1. ¿En qué orden aparecen los mensajes?

El orden que se ejecutan es como está el código programado

1. ¿Qué ocurriría si una línea tarda mucho tiempo en ejecutarse?

En JavaScript el código síncrono bloquea el hilo principal, lo que indica que es secuencial, por lo tanto si una línea se tara mucho en ejecutarse el resto espera a que finalice.

1. ¿Qué pasaría con la experiencia del usuario si el código se queda “bloqueado”?

El primer pensamiento es que el programa no funciona, que bloqueo el navegador, por esta razón es mejor sean asíncronas para que esto no se vea, no es mal programación ni funcionamiento, dependerá del recurso donde se ejecuta.

**Parte III: Sincronía con setTimeout**

Teoría: setTimeout permite posponer la ejecución de una función después de un tiempo determinado, sin bloquear el resto del código.

Práctica: En el mismo archivo, escribe y ejecuta el siguiente código:

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

1. ¿Por qué aparece “Fin” antes que el mensaje de setTimeout?

Node.js al encontrar un setTimeout lo envía a un temporizador, para ejecutar el resto de las instrucciones, por eso salen antes del mensaje de los 2 segundos.

1. ¿El programa se detuvo 2 segundos o siguió ejecutándose?

El programa continúa ejecutándose, en ningún momento se pausa, solo deja programado el callback y mide el tiempo de los 2 segundos que se indican.

1. ¿Cómo ayuda esto en aplicaciones interactivas?

Permite no bloquear el resto del programa, mientras lee, procesa o realiza cálculos, con esto la aplicación puede tender otras peticiones y el usuario no ve bloqueo en sus respuestas.

**Parte IV: Callbacks**

Teoría: Un callback es una función que se pasa como argumento a otra función, y se ejecuta cuando la tarea termina.

Práctica: En tu archivo de prueba, escribe y ejecuta el siguiente código:

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

1. ¿Qué diferencias notas con el callback?

Con callbacks debe existir otra función, la cual promesas devuelve un objeto promise y el .then lo hace después; el código de promesas es mas práctico y entendible por la estructura que tiene

1. ¿Dónde se especifica qué hacer después de que se descarga el archivo?

En el bloque de la función donde esta el .then() y se ejcuta cuando la promesa finaliza

1. ¿Qué ventaja tiene encadenar .then()?
2. Hace los procesos asíncronos mas ordenado y evita muchos niveles en el programa, permitiendo ejecutarse en orden.

**Parte V: Promesas**

Teoría: Una promesa representa un valor que estará disponible ahora, más tarde o nunca. Evita la llamada “callback hell”.

Práctica: En tu archivo de “experimentos” (estamos en un laboratorio ¿verdad?) escribe y ejecuta el siguiente código:

***Texto

El contenido generado por IA puede ser incorrecto.***

1. ¿Qué diferencias notas con el callback?

Con callback la función tendría que recibir otra función como parámetro para que se ejecute al terminar, con promesa devuelve un objeto Promise que lo ejecutará después, volvineo mas estructuradlo el código y más legible.

1. ¿Dónde se especifica qué hacer después de que se descarga el archivo?

En el bloque donde está el .then()

1. ¿Qué ventaja tiene encadenar .then()?

Sirve para ejecutar pasos en orden y esperando que finalice el anterior.

***Parte VI***

Teoría: async/await hace que el código asincrónico se lea de forma secuencial y clara, como si fuera síncrono, pero sin bloquear.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

1. ¿Por qué parece que el código se ejecuta línea por línea, aunque es asincrónico?

Porque la función await hace la pausa con la función async, esperando finalice promesa, el programa continua su ejecución

1. ¿Qué ventajas tiene frente a promesas con .then()?

El código es más fácil de leer e interpretar, se puede adiconar manejo de error fácilmente yno se utiliza .then()

1. ¿Cuándo preferirías usar async/await en un proyecto real?

Cuando son muchas funciones u operaciones asíncronas y son precedentes; esto es de mucha utilidad en el Backend, sin embargo el .thne() se puede utilizar para proceso aislados y/o que hay varios proceso en paralelo ejecutándose.

Link GitGub

<https://github.com/jnelsonmenjivarg/PED_Laboratorios.git>