



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	<p>Entrega final: <u>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.</u></p> <p>Actividades a realizar:</p> <p>Parte I: Configurando el entorno</p> <p>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.</p> <p>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.</p> <p>Parte II: Hablemos de Sincronía</p> <p>Teoría: En un programa síncrono, las instrucciones se ejecutan línea por línea, esperando a que cada una termine antes de continuar.</p> <p>Práctica: Crea una carpeta y ábrela desde VS Code. Crea un archivo llamado asincronia.js Escribe y ejecuta el siguiente código (abre una terminal en vs code y ejecuta el comando node asincronia.js):</p> <div><pre>console.log("Inicio"); console.log("Proceso..."); console.log("Fin");</pre></div>

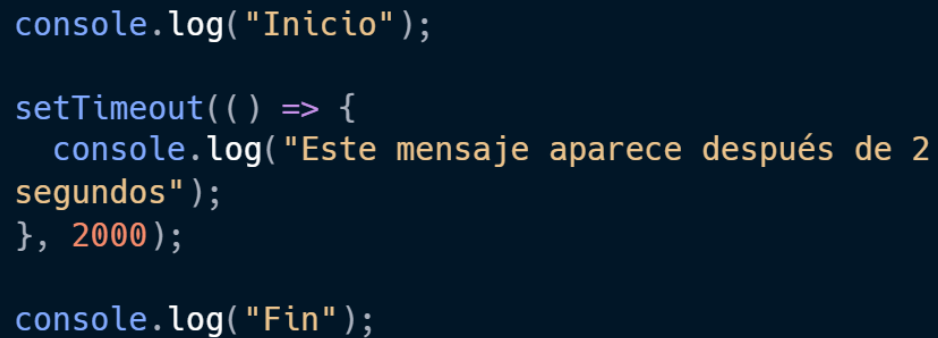
Responde a las siguientes preguntas:

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:



```
console.log("Inicio");

setTimeout(() => {
  console.log("Este mensaje aparece después de 2 segundos");
}, 2000);

console.log("Fin");
```


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:



```
function descargarArchivo(nombre, callback) {  
  console.log(`Descargando ${nombre}...`);  
  setTimeout(() => {  
    console.log(`${nombre} descargado`);  
    callback(); // se ejecuta cuando termina  
  }, 2000);  
}  
  
descargarArchivo("archivo1.txt", () => {  
  console.log("Procesando archivo descargado...");  
});
```


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:



```
function descargarArchivo(nombre) {
  return new Promise((resolve) => {
    console.log(`Descargando ${nombre}...`);
    setTimeout(() => {
      console.log(`${nombre} descargado`);
      resolve(nombre);
    }, 2000);
  });
}

descargarArchivo("archivo2.txt")
  .then((resultado) => {
    console.log(`Procesando ${resultado}...`);
  });
```

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 asíncrono se lea de forma secuencial y clara, como si fuera síncrono, pero sin bloquear.

Práctica: Escribe y ejecuta el siguiente código:



```
async function main() {
  const resultado = await
  descargarArchivo("archivo3.txt");
  console.log(`📁 Procesando ${resultado}...`);
}

main();
```

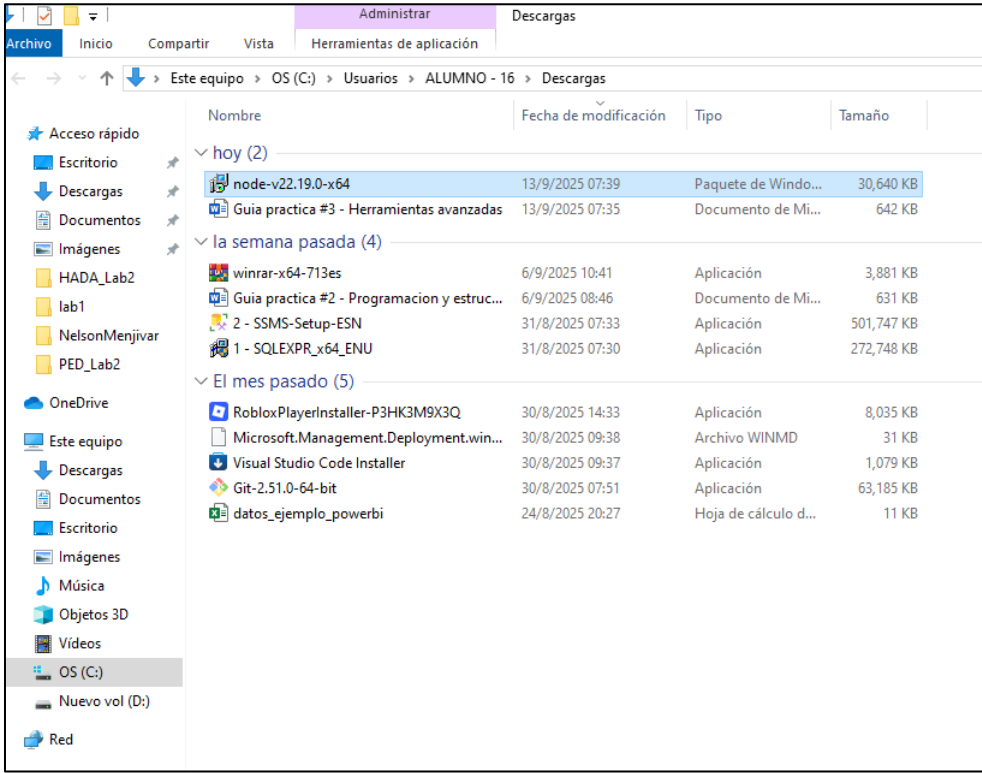
Responde a las siguientes preguntas:

1. ¿Por qué parece que el código se ejecuta línea por línea aunque es asíncrono?
2. ¿Qué ventajas tiene frente a promesas con `.then()`?

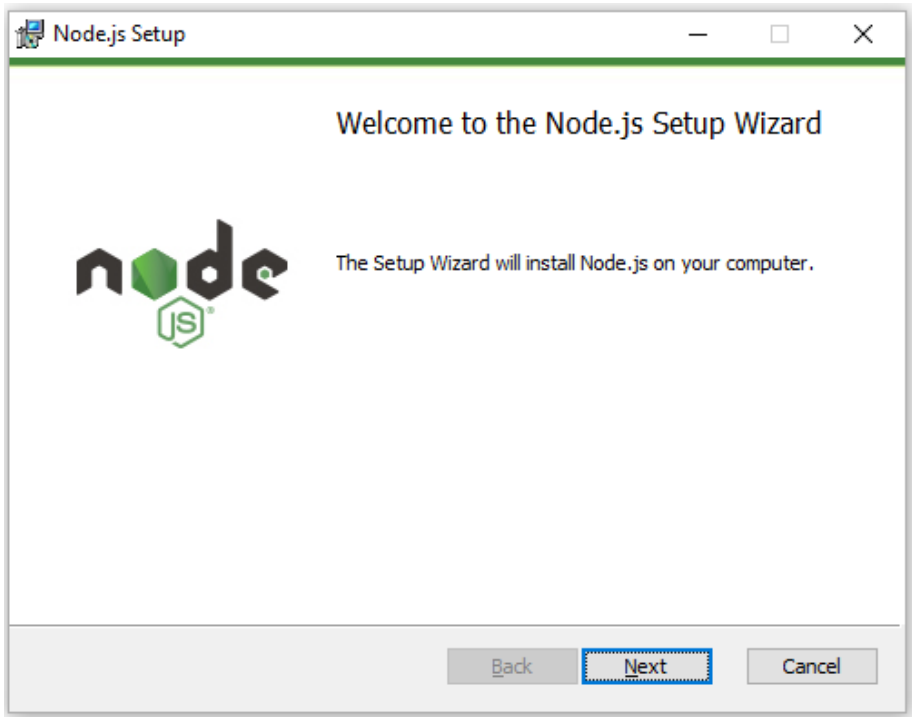
	3. ¿Cuándo preferirías usar async/await en un proyecto real?
Ponderación	12.5% de la quinta nota

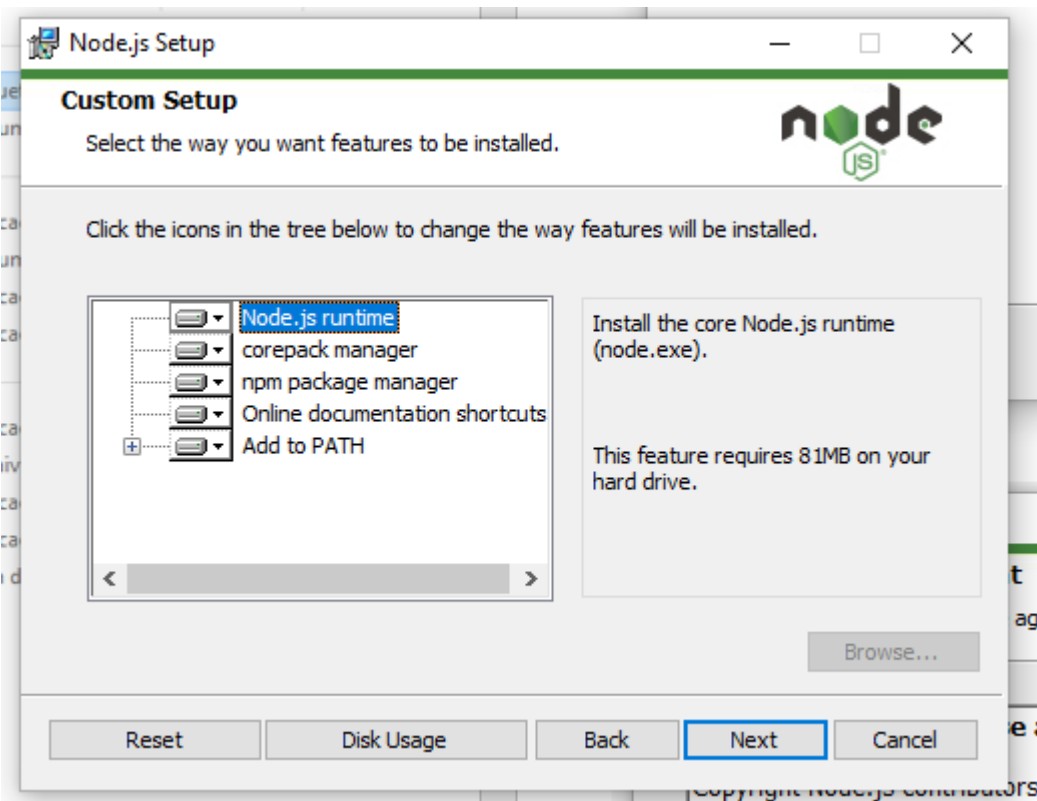
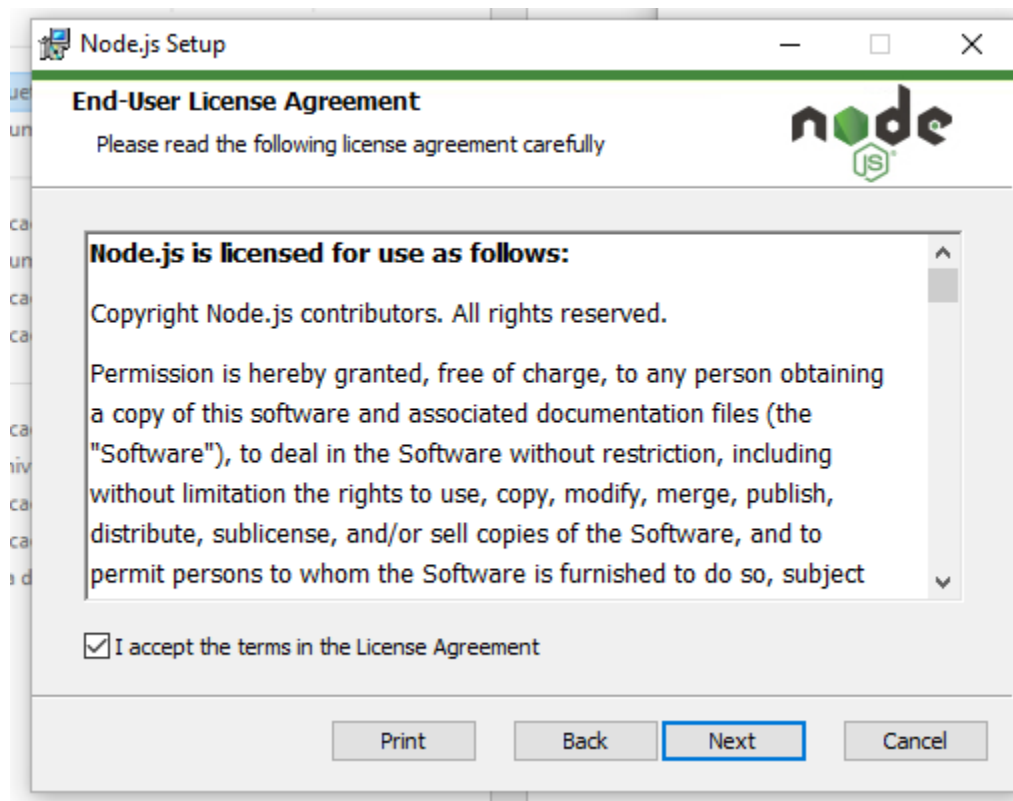
Solución a Laboratorio:

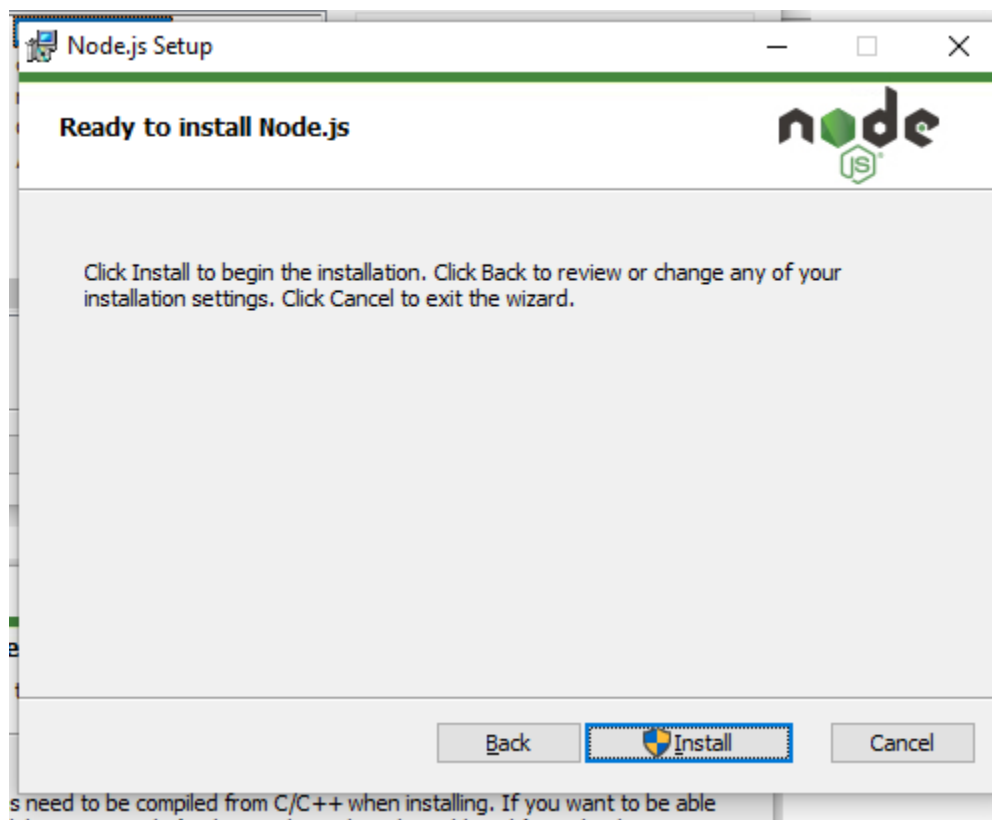
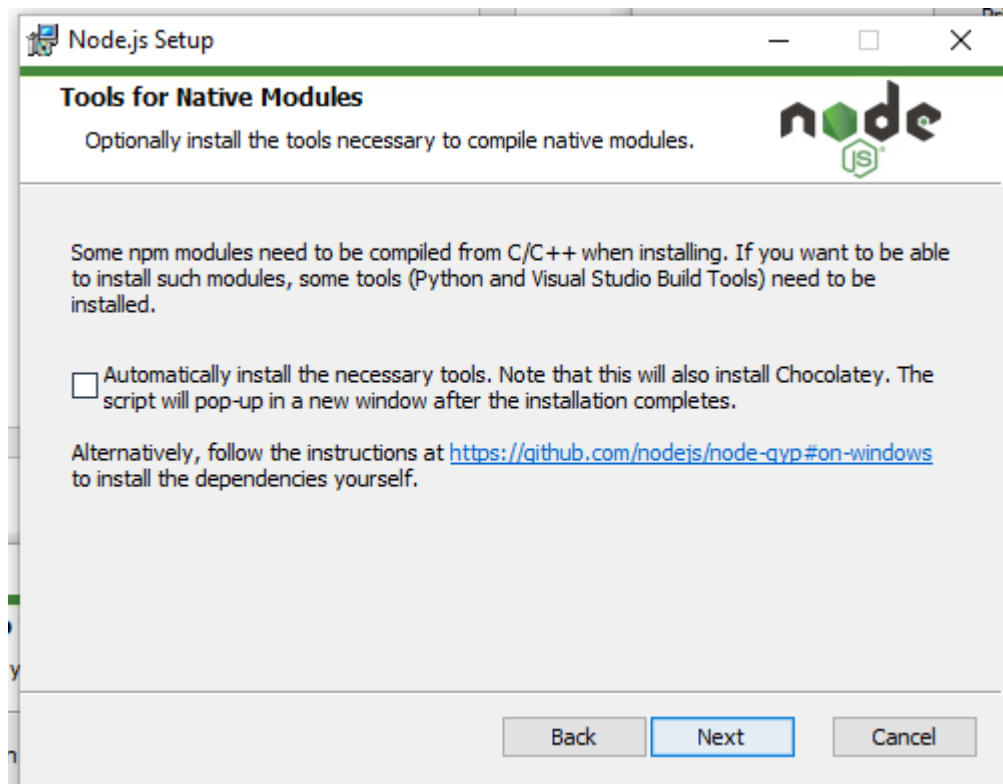
Descarga de Node

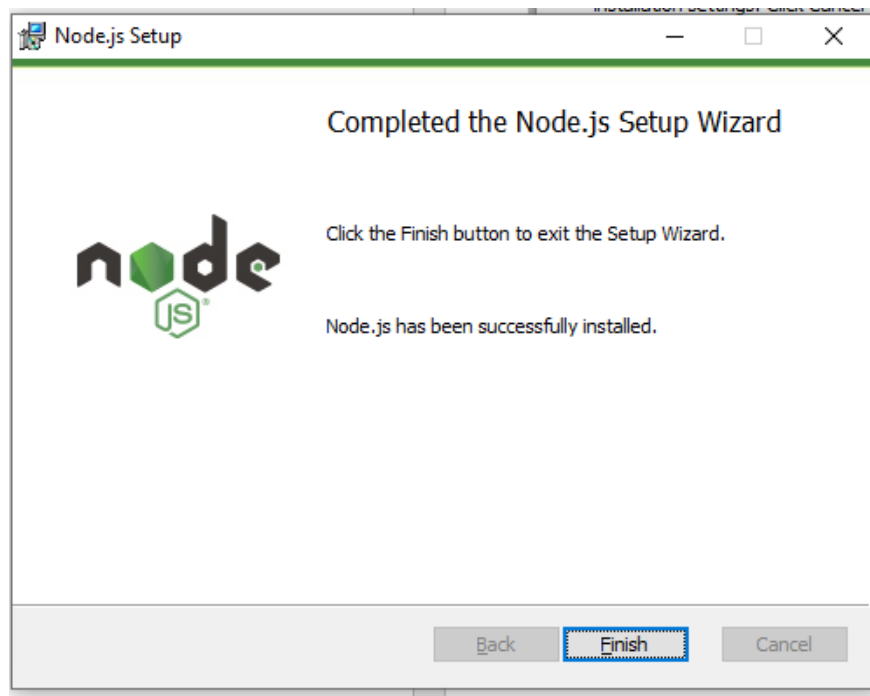


Instalacion









```
Símbolo del sistema
Microsoft Windows [Versión 10.0.19045.6216]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\ALUMNO - 16>node -v
v22.19.0

C:\Users\ALUMNO - 16>
```

```
Símbolo del sistema
v22.19.0

C:\Users\ALUMNO - 16>cd\

C:\> cd NelsonMenjivar

C:\NelsonMenjivar>dir
El volumen de la unidad C es OS
El número de serie del volumen es: F0E5-3D03

Directorio de C:\NelsonMenjivar

13/09/2025 07:28 <DIR>      .
13/09/2025 07:28 <DIR>      ..
06/09/2025 10:36 <DIR>      HADA_Lab2
13/09/2025 07:29 <DIR>      HADA_Lab3
30/08/2025 10:47      1,203,640 Lab 1 HADA 30082025.docx
06/09/2025 10:39      6,846,234 Lab 1 HADA 30082025.zip
30/08/2025 10:16      811,169 Lab 1 PED 30082025.docx
06/09/2025 10:46      4,347,429 Lab2NelsonMenjivar.rar
06/09/2025 10:42      13,079,860 Laboratorio2.rar
06/09/2025 10:40 <DIR>      PED_Lab2
                    5 archivos    26,288,332 bytes
                    5 dirs    146,329,432,064 bytes libres

C:\NelsonMenjivar>cd HADA_Lab3

C:\NelsonMenjivar\HADA_Lab3>code .

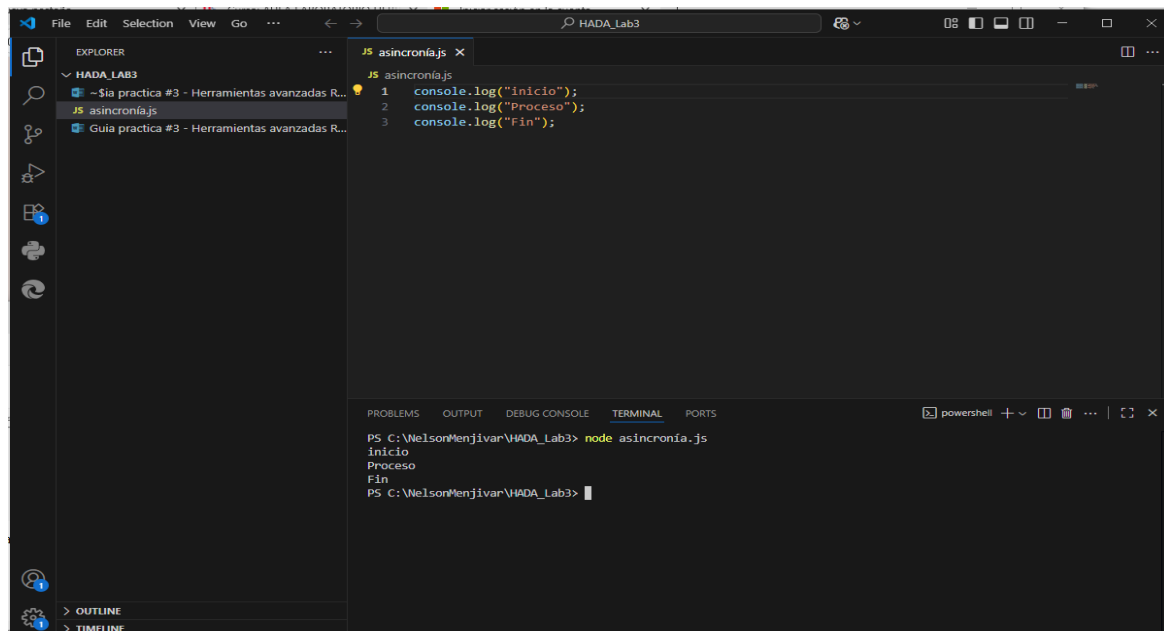
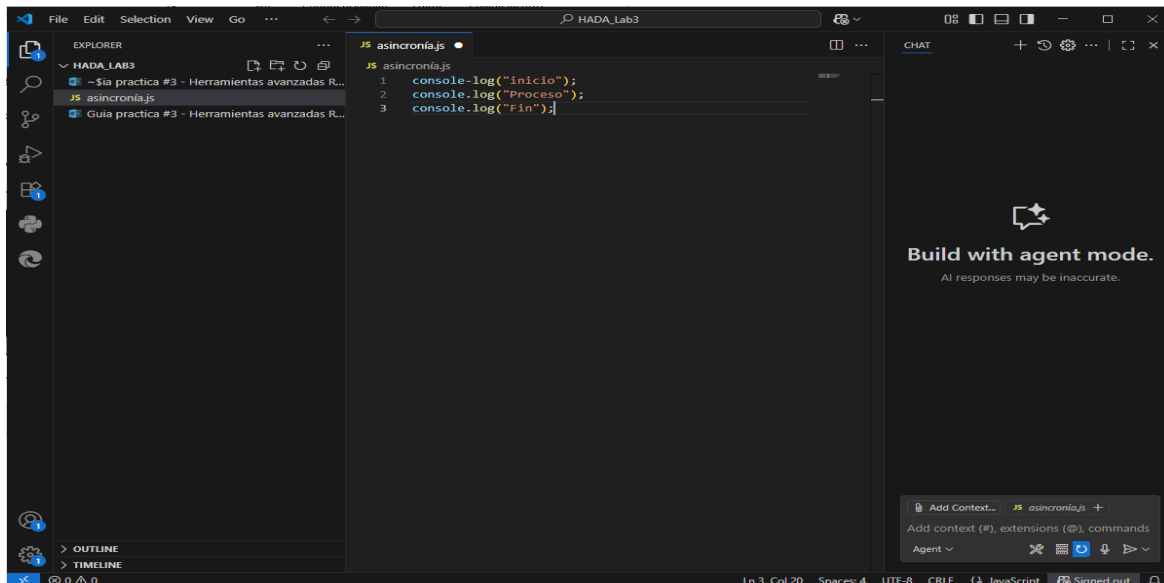
C:\NelsonMenjivar\HADA_Lab3>
```


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`):



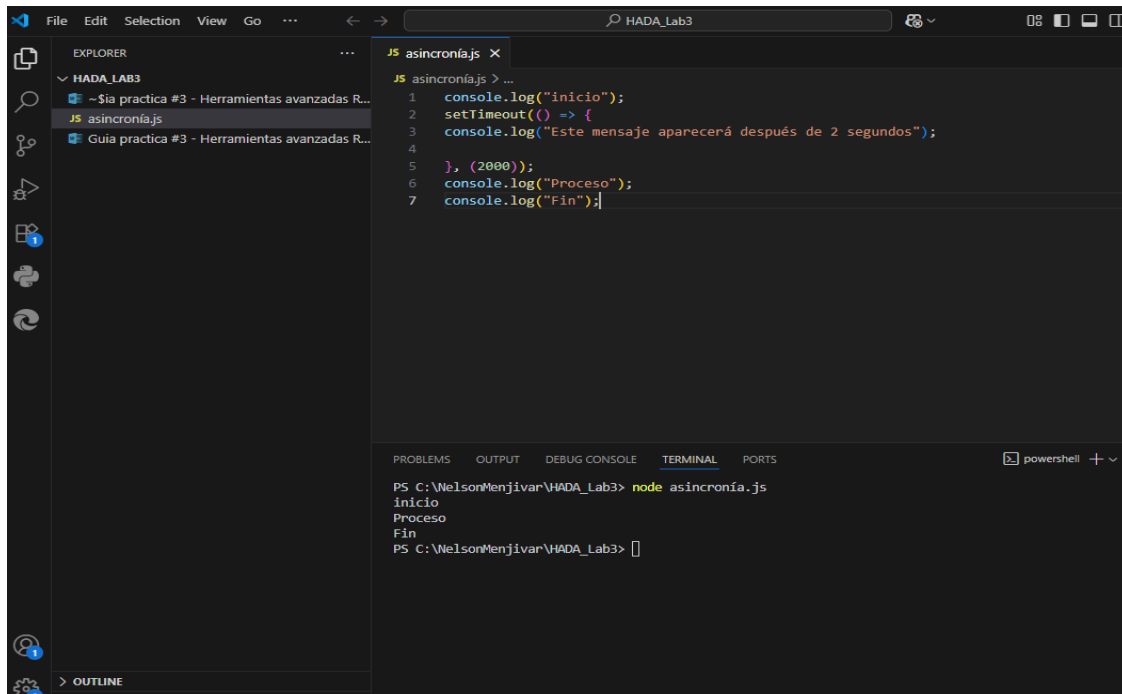
1. ¿En qué orden aparecen los mensajes?
El orden que se ejecutan es como está el código programado
2. ¿Qué ocurriría si una línea tarda mucho tiempo en ejecutarse?
En JavaScript el código síncrono bloquea el hilo principal, lo que indica que es secuencial, por lo tanto si una línea se tarta mucho en ejecutarse el resto espera a que finalice.
3. ¿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.

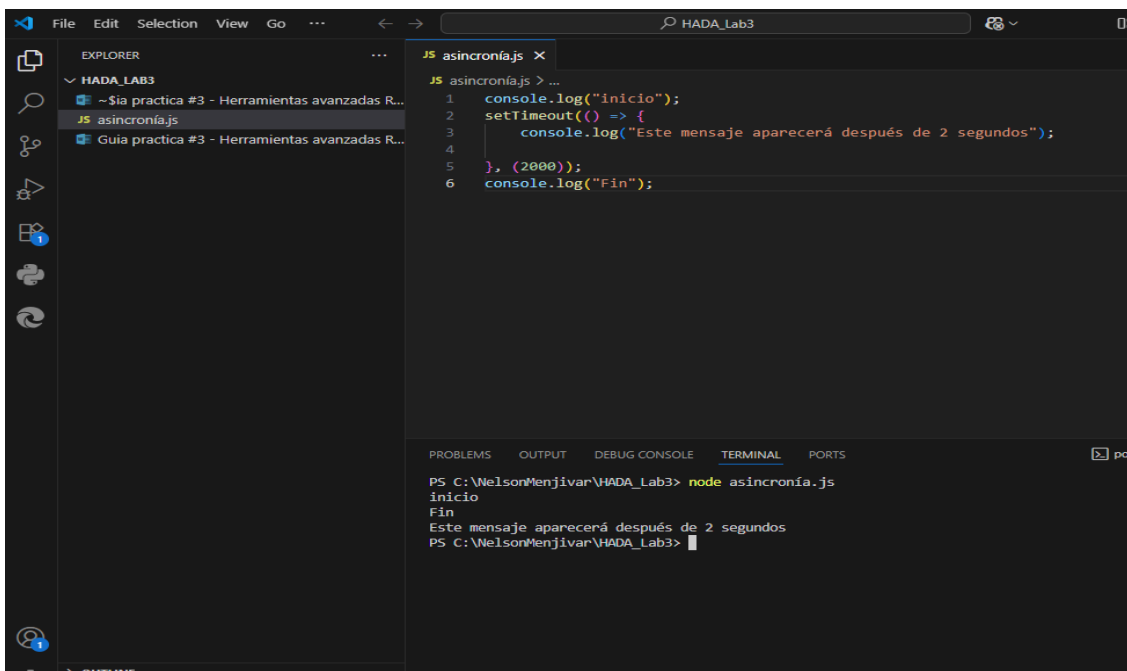
Parte III: Sincronía con **setTimeout**

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:



```
File Edit Selection View Go ... HADA_Lab3
EXPLORER
  HADA_LAB3
    ~$ia practica #3 - Herramientas avanzadas R...
    JS asincronía.js
    Guia practica #3 - Herramientas avanzadas R...
  JS asincronía.js
    1 console.log("inicio");
    2 setTimeout(() => {
    3   console.log("Este mensaje aparecerá después de 2 segundos");
    4 }, (2000));
    5 console.log("Proceso");
    6 console.log("Fin");
    7
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\WilsonMenjivar\HADA_Lab3> node asincronía.js
inicio
Proceso
Fin
PS C:\WilsonMenjivar\HADA_Lab3>
```



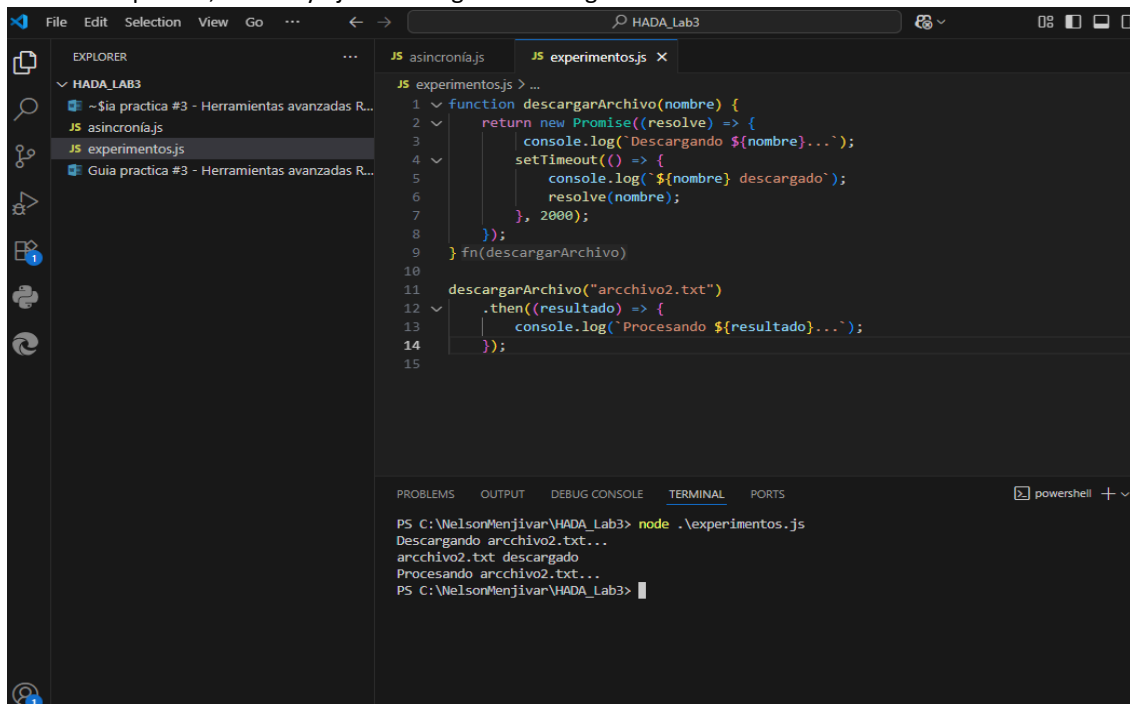
```
File Edit Selection View Go ... HADA_Lab3
EXPLORER
  HADA_LAB3
    ~$ia practica #3 - Herramientas avanzadas R...
    JS asincronía.js
    Guia practica #3 - Herramientas avanzadas R...
  JS asincronía.js
    1 console.log("inicio");
    2 setTimeout(() => {
    3   console.log("Este mensaje aparecerá después de 2 segundos");
    4 }, (2000));
    5 console.log("Fin");
    6
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\WilsonMenjivar\HADA_Lab3> node asincronía.js
inicio
Fin
Este mensaje aparecerá después de 2 segundos
PS C:\WilsonMenjivar\HADA_Lab3>
```

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.
2. ¿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.
3. ¿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:



```
File Edit Selection View Go ... HADA_Lab3
EXPLORER
HADA_LAB3
  ~$ia practica #3 - Herramientas avanzadas R...
  JS asincronia.js
  JS experimentos.js
  Guia practica #3 - Herramientas avanzadas R...

JS experimentos.js
1  function descargarArchivo(nombre) {
2    return new Promise((resolve) => {
3      console.log(`Descargando ${nombre}...`);
4      setTimeout(() => {
5        console.log(`${nombre} descargado`);
6        resolve(nombre);
7      }, 2000);
8    });
9  }
10
11  descargarArchivo("archivo2.txt")
12    .then((resultado) => {
13      console.log(`Procesando ${resultado}...`);
14    });
15

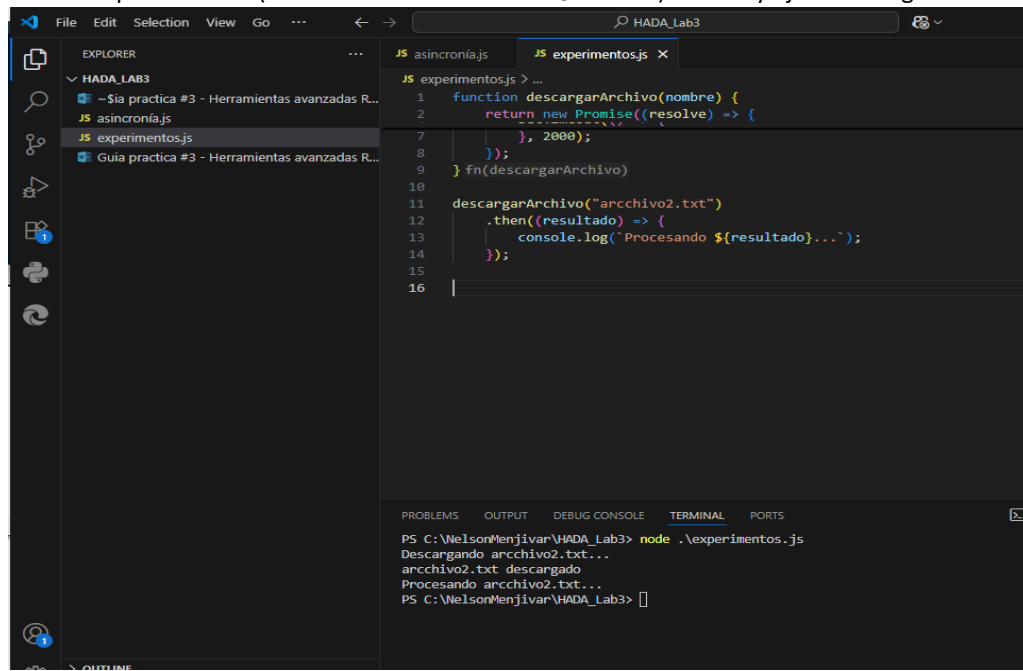
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\WilsonMenjivar\HADA_Lab3> node .\experimentos.js
Descargando archivo2.txt...
archivo2.txt descargado
Procesando archivo2.txt...
PS C:\WilsonMenjivar\HADA_Lab3>
```

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
2. ¿Dónde se especifica qué hacer después de que se descarga el archivo?
En el bloque de la función donde esta el .then() y se ejecuta cuando la promesa finaliza
3. ¿Qué ventaja tiene encadenar .then()?
4. 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:



```
File Edit Selection View Go ... < > HADA_Lab3
EXPLORER
  HADA_LAB3
    ~$ia practica #3 - Herramientas avanzadas R...
    JS asincronia.js
    JS experimentos.js
    Guia practica #3 - Herramientas avanzadas R...

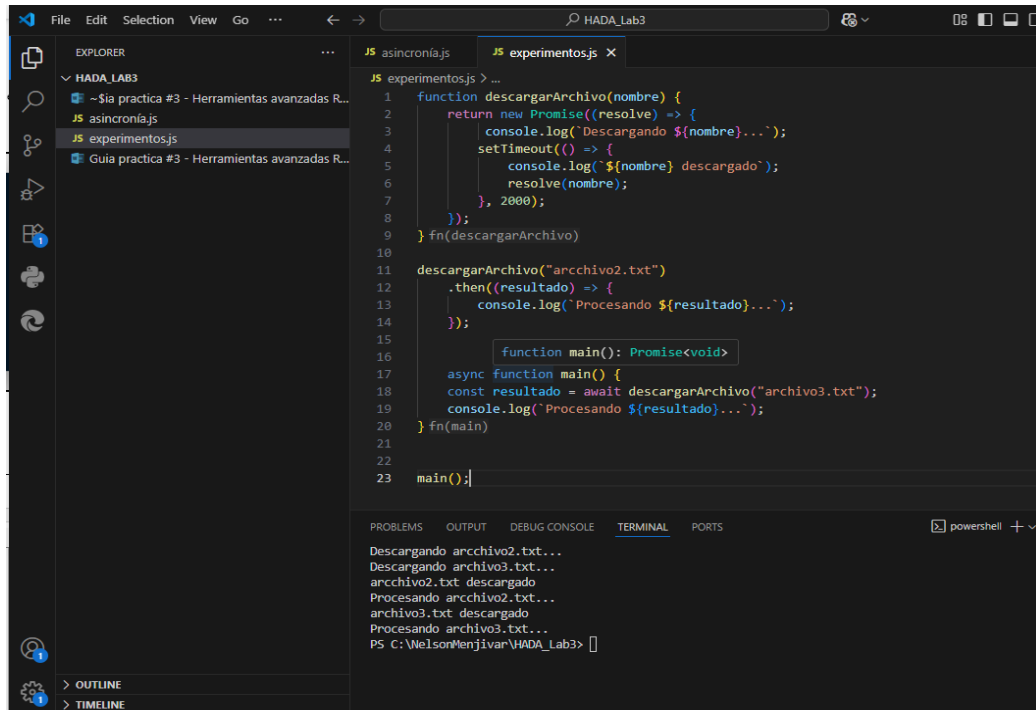
JS experimentos.js
1  function descargarArchivo(nombre) {
2      return new Promise((resolve) => {
3          // ...
4          // ...
5          // ...
6          // ...
7      });
8  }
9  fn(descargarArchivo)
10
11  descargarArchivo("archivo2.txt")
12      .then((resultado) => {
13          console.log('Procesando ${resultado}...');
14      });
15
16

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\NelsonMenjivar\HADA_Lab3> node .\experimentos.js
Descargando archivo2.txt...
archivo2.txt descargado
Procesando archivo2.txt...
PS C:\NelsonMenjivar\HADA_Lab3>
```

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, volviendo más estructurado el código y más legible.
2. ¿Dónde se especifica qué hacer después de que se descarga el archivo?
En el bloque donde está el .then()
3. ¿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 asíncrono se lea de forma secuencial y clara, como si fuera síncrono, pero sin bloquear.



The screenshot shows a Visual Studio Code editor with a file explorer on the left and a code editor on the right. The code editor displays a JavaScript file named 'experimentos.js'. The code defines a function 'descargarArchivo' that returns a Promise, and a 'main' function that uses 'async' and 'await' to call 'descargarArchivo'. The terminal at the bottom shows the output of the code execution, which is sequential despite the asynchronous nature of the functions.

```
1 function descargarArchivo(nombre) {
2   return new Promise((resolve) => {
3     console.log("Descargando ${nombre}...");
4     setTimeout(() => {
5       console.log(`${nombre} descargado`);
6       resolve(nombre);
7     }, 2000);
8   });
9 }
10
11 descargarArchivo("archivo2.txt")
12   .then((resultado) => {
13     console.log("Procesando ${resultado}...");
14   });
15
16 function main(): Promise<void>
17 async function main() {
18   const resultado = await descargarArchivo("archivo3.txt");
19   console.log("Procesando ${resultado}...");
20 }
21
22
23 main();
```

Terminal Output:

```
Descargando archivo2.txt...
Descargando archivo3.txt...
archivo2.txt descargado
Procesando archivo2.txt...
archivo3.txt descargado
Procesando archivo3.txt...
PS C:\NelsonMenjivar\HADA_Lab3>
```

1. ¿Por qué parece que el código se ejecuta línea por línea, aunque es asíncrono?
Porque la función await hace la pausa con la función async, esperando finalice promesa, el programa continua su ejecución
2. ¿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()
3. ¿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