

Explicación detallada del código asíncrono en JavaScript

1. La función esperar(ms)

```
javascript Copiar Editar  
  
function esperar(ms) {  
  return new Promise(resolve => setTimeout(resolve, ms));  
}
```

¿Qué es una promesa?

Una **promesa** (Promise) es un objeto en JavaScript que representa un valor que puede no estar una forma de manejar operaciones que tomarán algo de tiempo, como la lectura de archivos o una solicitud a un servidor. disponible en el momento, pero estará en el futuro. Básicamente, una promesa es

- **new Promise()** crea una nueva promesa.
- **setTimeout(resolve, ms)** establece un temporizador que esperará **ms** milisegundos y luego ejecutará el código que está dentro de la función de resolución (en este caso, **resolve**).
- **resolve**: Es una función que marca que la tarea asíncrona ha terminado correctamente.

Entonces, cuando usamos **esperar(2000)**, creamos una promesa que se resolverá después de 2000 milisegundos (2 segundos).

2. La función lavarFruta(fruta)

```
javascript Copiar Editar  
  
async function lavarFruta(fruta) {  
  console.log(`Lavando la ${fruta}...`);  
  await esperar(2000);  
  console.log(`${fruta} lavada`);  
  return fruta;  
}
```

Uso de `async` y `await`:

- **`async`**: Se usa para marcar que la función contiene código asíncrono. Esto significa que la función devolverá una promesa, incluso si no usamos explícitamente una promesa dentro de la función.
- **`await`**: Se usa dentro de funciones **`async`** para hacer una "pausa" hasta que la promesa se resuelva. En lugar de escribir `.then()` para manejar una promesa, **`await`** hace que el código espere hasta que la promesa se resuelva o se rechace.

En esta función, estamos haciendo lo siguiente:

1. Imprimimos el mensaje "Lavando la fruta...".
2. Llamamos a **`await esperar(2000)`**, lo que hace que el código se "pause" por 2 segundos.
3. Después de esos 2 segundos, imprimimos "Fruta lavada".
4. Finalmente, devolvemos el nombre de la fruta.

Lo interesante es que, aunque el código de la función parece ejecutarse de manera secuencial, en realidad, está manejando la espera de forma **asíncrona**. Eso significa que durante esos 2 segundos, el programa puede seguir haciendo otras cosas.

3. La función `licuarFruta(fruta)`


```
javascript Copiar código  
  
async function licuarFruta(fruta) {  
  console.log(`Licuando la fruta ${fruta}...`);  
  await esperar(3000);  
  const jugo = `jugo de ${fruta}`;  
  console.log(`${fruta} listo`);  
  return jugo;  
}
```

Esta función es similar a la anterior:

1. Imprime "Licuando la fruta...".
2. Luego, espera 3 segundos (**`await esperar(3000)`**).
3. Después de los 3 segundos, crea una variable `jugo` con el nombre del jugo (por ejemplo, "jugo de fresa").
4. Imprime "Fruta lista", y finalmente, devuelve el jugo.

4. La función servirJugo(jugo)

javascript

 Copiar código


```
async function servirJugo(jugo) {  
  console.log(`Sirviendo el ${jugo}...`);  
  await esperar(1000);  
  console.log(`${jugo} servido. ¡A disfrutar!`);  
}
```

Copiar

Esta función también sigue el mismo patrón. Sirve el jugo y espera 1 segundo antes de imprimir que el jugo está servido.

5. La función prepararJugo()

javascript

 Copiar código

```
async function prepararJugo() {  
  console.log("Preparando un delicioso jugo...\n");  
  const frutaLavada = await lavarFruta("fresa");  
  const jugoHecho = await licuarFruta(frutaLavada);  
  await servirJugo(jugoHecho);  
  console.log("\nJugo terminado. ¡Salud!");  
}
```


Explicación:

Esta es la función principal que coordina todas las otras funciones. El flujo es el siguiente:

1. Primero, imprime "Preparando un delicioso jugo...".
2. Luego, llama a **await lavarFruta("fresa")**, lo que significa que esperará a que la fruta esté lavada antes de continuar. El resultado (**frutaLavada**) es la fruta que se lavó.
3. Luego, pasa la fruta lavada a **licuarFruta(frutaLavada)**. Este paso espera a que la fruta sea licuada y guarda el resultado en **jugoHecho**.
4. Después, pasa el jugo a **servirJugo(jugoHecho)** para que lo sirvan. También espera hasta que esta acción se complete.
5. Finalmente, imprime "Jugo terminado. ¡Salud!".

6. Ejecución de prepararJugo()

javascript

 Copiar código

```
prepararJugo();
```

Cuando llamamos a **prepararJugo()**, comienza el proceso de hacer el jugo de fresa. Como estamos utilizando **await**, el flujo de ejecución sigue el orden de las operaciones que definimos, esperando que cada tarea asíncronica se complete antes de pasar a la siguiente.