ThePower business school

Async & Await JS 🥶🤖

`async` y `await` son palabras clave de JavaScript que se utilizan para trabajar con promesas de forma más sencilla y clara.

La palabra clave `async` se utiliza para declarar una función asíncrona. Una función asíncrona es una función que puede contener código asíncrono y devuelve una promesa. La palabra clave `async` se coloca antes de la declaración de la función:

```
async function miFuncionAsincrona() {
  // código asíncrono va aquí
}
```

La palabra clave `await` se utiliza dentro de una función asíncrona para esperar a que una promesa se cumpla y obtener su valor. Cuando se encuentra `await`, la ejecución de la función se detiene hasta que se cumple la promesa. Luego, se continúa la ejecución de la función y se obtiene el valor de la promesa.

Aquí hay un ejemplo de cómo se puede utilizar `await` dentro de una función asíncrona:

```
async function obtenerDatos() {
  const respuesta = await fetch('https://example.com/api/endpoint');
  const datos = await respuesta.json();
  console.log(datos);
}
```

En este ejemplo, primero utilizamos `fetch` para realizar una solicitud HTTP y obtener una promesa. Luego, utilizamos `await` para esperar a que se cumpla la promesa y obtener el objeto `Response`. Finalmente, utilizamos `await` de nuevo para esperar a que se cumpla la promesa del método `json` y obtener los datos en formato JSON.

Uso de async / await

Cuando se declara una función **async** convierte al elemento en una promesa, la cual se resolverá mediante el retorno. En estas funciones podemos hacer uso de la expresión **await** para pausar la ejecución a esperas de la resolución de una promesa, la cual se reanudará una vez cumplida pudiendo manejar la asincronía y resolución de **promesas**.

```
//Con promesas

function getInfo() {
  fetch(URL).then((raw) => {
    raw.json().then((formatted) => {
      return formatted;
    });
  });
}

//Con async/await

async function getInfo() {
  const raw = await fetch(URL);
  const formatted = await raw.json();
  return formatted;
}
```

En este ejemplo vemos como la ejecución de pausa hasta completar por completo una petición mediante el **fetch** y no reanudará hasta estar completada. De igual manera la ejecución se pausa al formatear los datos obtenidos en la petición hasta que estén convertidos a JSON. Una vez tratadas estas dos líneas de manera asíncrona, la función retorna el resultado.

En otras palabras, gracias al tener una función asíncrona mediante la expresión **async** podemos hacer que los bloques de ejecución se completen sin que toda la ejecución de la función caiga en cascada sin pausa, ejecutando una a una y esperando a su finalización las líneas indicadas con la expresión **await**.

En el caso que utilicemos **arrow functions:**

```
const getInfo = async() => {
  const raw = await fetch(URL);
  [...]
}
```

petición con async/await

se puede utilizar `async` y `await` para realizar una solicitud HTTP a la PokeAPI y obtener información sobre un Pokémon específico de forma más sencilla y clara.

Aquí hay un ejemplo de cómo se puede hacer esto:

```
async function obtenerPokemon(pokemonName) {
  try {
    const respuesta = await fetch(`https://pokeapi.co/api/v2/pokemon/${pokemonName}`);
    const datos = await respuesta.json();
    console.log(datos);
  } catch (error) {
    console.error(error);
  }
}

obtenerPokemon('pikachu'); // imprime información sobre el Pokémon Pikachu


const obtenerPokemon = async (pokemonName) => {
  try {
    const respuesta = await fetch(`https://pokeapi.co/api/v2/pokemon/${pokemonName}`);
    const datos = await respuesta.json();
    console.log(datos);
  } catch (error) {
    console.error(error);
  }
};

obtenerPokemon('pikachu'); // imprime información sobre el Pokémon Pikachu
```

En este ejemplo, hemos creado una función asíncrona `obtenerPokemon` que toma un nombre de Pokémon como argumento y realiza una solicitud HTTP `GET` a la URL de la PokeAPI para ese Pokémon. Luego, utilizamos `await` para esperar a que se cumpla la promesa de la solicitud y obtener el objeto `Response`, y utilizamos `await` de nuevo para esperar a que se cumpla la promesa del método `json` y obtener los datos en formato JSON.

Pintando en el DOM

Para mostrar la información sobre un Pokémon en el DOM (Document Object Model), primero necesitarías obtener esa información utilizando `fetch` o `async`/`await`. Una vez que tienes los datos, puedes utilizar JavaScript para crear elementos del DOM y agregarlos a la página para mostrar la información.

Aquí hay un ejemplo de cómo se puede hacer esto utilizando `async`/`await` y arrow functions:

```
const mostrarPokemon = async (pokemonName) => {
  try {
    const respuesta = await fetch(`https://pokeapi.co/api/v2/pokemon/${pokemonName}`);
    const datos = await respuesta.json();

    // crear elementos del DOM y establecer sus atributos y contenido
    const nombre = document.createElement('h1');
    nombre.textContent = datos.name;

    const imagen = document.createElement('img');
    imagen.src = datos.sprites.front_default;

    // agregar elementos al DOM
    document.body.appendChild(nombre);
    document.body.appendChild(imagen);
  } catch (error) {
    console.error(error);
  }
};

mostrarPokemon('pikachu'); // muestra información sobre el Pokémon Pikachu en la página
```

En este ejemplo, primero utilizamos `fetch` y `async`/`await` para obtener la información sobre el Pokémon específico de la PokeAPI. Luego, creamos elementos del DOM como `h1` y `img` y establecemos sus atributos y contenido utilizando las propiedades de los datos obtenidos de la PokeAPI. Finalmente, agregamos estos elementos al DOM utilizando el método `appendChild` del objeto `document.body`.