In JavaScript, **Promises** and **Async/Await** are both methods for handling asynchronous operations. Here's a breakdown of both:

1. Promises

A Promise is an object that represents a future value. It may be resolved or rejected after an asynchronous operation completes. Promises are useful when chaining multiple asynchronous operations together.

Syntax:

```javascript
const promise = new Promise((resolve, reject) => {
  // async operation
  if (success) resolve(result);
  else reject(error);
});

promise
  .then(result => console.log(result)) // handle success
  .catch(error => console.error(error)); // handle error
```

Usage:

- Promises are used when you want to handle asynchronous operations (like API calls, reading files, etc.).
- You can chain multiple `.then()` methods to handle sequential asynchronous operations.
- `.catch()` is used to handle errors in the chain of promises.

Example:

```javascript
const asyncOperation = new Promise((resolve, reject) => {
  setTimeout(() => {
    const success = true;
    if (success) resolve('Operation successful!');
    else reject('Operation failed.');
  }, 1000);
});

asyncOperation
  .then(result => console.log(result)) // Logs: 'Operation successful!'
  .catch(error => console.error(error)); // In case of failure
```

2. Async/Await

`async/await` is built on top of Promises but provides a more synchronous and readable way to handle asynchronous code. The `await` keyword can only be used inside an `async` function, and it pauses the execution until the Promise is resolved or rejected.

Syntax:

```javascript
async function fetchData() {
  try {
    const response = await fetch('https://api.example.com/data'); // Wait for
the fetch Promise to resolve
    const data = await response.json(); // Wait for the response to be parsed
    console.log(data); // Handle the data
  } catch (error) {
    console.error(error); // Handle the error
  }
}
```

Usage:

- Provides a cleaner and more readable way to write asynchronous code compared to chaining `.then()` and `.catch()`.
- Inside an `async` function, you can use `await` to pause the execution until the Promise is fulfilled, making the code look more synchronous.

Example:

```javascript
async function getUserData() {
  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/users');
    const users = await response.json();
    console.log(users);
  } catch (error) {
    console.error('Error fetching user data:', error);
  }
}

getUserData();
```

Key Differences:

- **Promises** use chaining with `.then()` and `.catch()`, which can lead to a "callback-like" structure.
- **Async/Await** makes the code look more linear and synchronous, which improves readability.