I can provide you with an example of asynchronous functions in JavaScript. Asynchronous functions (also known as `async` functions) are functions that can contain asynchronous operations and use the `await` keyword to wait for the completion of a Promise. Here's an example:

```javascript
// Define an async function
async function fetchData() {
    // Simulating a fetch request to a URL
    const url = 'https://jsonplaceholder.typicode.com/todos/1';

    // Use the await keyword to wait for the fetch Promise to resolve
    const response = await fetch(url);

    // Once the Promise resolves, use the await keyword to parse the response as JSON
    const data = await response.json();

    // Log the data to the console
    console.log(data);
}

// Call the async function
fetchData();
```

In this example, the `fetchData` function is defined as an async function using the `async` keyword. Inside the function, asynchronous operations (such as a network request using the `fetch` API) are handled using the `await` keyword. This allows the function to wait for the Promise returned by the `fetch` function to resolve and for the Promise returned by `response.json()` to resolve before continuing. The data is then logged to the console.

In addition to fetching data, you can use async functions in many other scenarios involving asynchronous operations, such as file I/O, timers, or interacting with APIs.

Here are a few more examples of async functions in JavaScript that demonstrate various use cases:

1. **Using `async` and `await` with Promise-based asynchronous operations**:
   This example shows how to perform multiple asynchronous operations using `async` and `await` with Promise-based functions.

   ```javascript
   // A function that simulates a delay using a Promise
   ```

```
function delay(ms) {
    return new Promise(resolve => setTimeout(resolve, ms));
}

// Define an async function that uses the delay function
async function asyncTask() {
    console.log('Starting task...');

    // Use the await keyword to wait for the delay
    await delay(2000);

    console.log('Task completed after 2 seconds.');
}

// Call the async function
asyncTask();
```

2. **Using async functions with error handling**:
   This example shows how to use `try` and `catch` blocks within an async function to handle errors.

```javascript
async function fetchData() {
    const url = 'https://jsonplaceholder.typicode.com/invalid-endpoint';

    try {
        // Attempt to fetch data from the API
        const response = await fetch(url);

        // Check if the response status is OK (200)
        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status}`);
        }

        // Parse the response as JSON
        const data = await response.json();

        // Log the data to the console
        console.log(data);
    } catch (error) {
        // Handle any errors that occur during the fetch or parsing
        console.error('Error fetching data:', error);
    }
```

```
}

// Call the async function
fetchData();
```

3. **Async function with concurrent operations**:
   This example demonstrates how you can run multiple asynchronous operations concurrently using `Promise.all` within an async function.

```javascript
async function fetchMultipleData() {
    // URLs to fetch data from
    const urls = [
        'https://jsonplaceholder.typicode.com/todos/1',
        'https://jsonplaceholder.typicode.com/todos/2',
        'https://jsonplaceholder.typicode.com/todos/3'
    ];

    try {
        // Use Promise.all to wait for all fetches to complete
        const responses = await Promise.all(urls.map(url => fetch(url)));

        // Parse the responses concurrently
        const data = await Promise.all(responses.map(response => response.json()));

        // Log the data to the console
        console.log(data);
    } catch (error) {
        // Handle any errors that occur during the fetch or parsing
        console.error('Error fetching data:', error);
    }
}

// Call the async function
fetchMultipleData();
```

In the last example, you can see how `Promise.all` is used to fetch data from multiple URLs concurrently. This allows the async function to wait for all promises to resolve before parsing the responses.