JavaScript uses the promise chaining mechanism to manage asynchronous operations. Promises are objects that reflect when an asynchronous operation will eventually succeed (or fail), and they let you attach callbacks to handle the outcomes or errors after the process is finished.

Promise chaining enables you to sequentially execute asynchronous actions while assuring that each job is dependent on the success of the prior one. As a result, the code structure is easier to comprehend and manage than with nested callbacks, commonly referred to as "callback hell."

Here's an example of promise chaining using JavaScript:

Let's assume we have three asynchronous functions: 'fetchUserData()', 'processUserData()', and 'saveUserData()'.And each function returns a promise

Code:

```
function fetchUserData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            const userData = { id: 1, name: "John Doe", age: 30 };
            resolve(userData);
        }, 1000);
    });
}


function processUserData(userData) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            userData.age++;
            resolve(userData);
        }, 500);
    });
}

function saveUserData(userData) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            console.log("User data saved:", userData);
```

```
            resolve();
        }, 800);
    });
}


fetchUserData()
    .then(processUserData)
    .then(saveUserData)
    .then(() => {
        console.log("All operations completed!");
    })
    .catch((error) => {
        console.error("An error occurred:", error);
    });
```

In the example above, ‘fetchUserData’ is called first, and once it completes successfully, the data is passed to ‘processUserData’. After processUserData is done, the processed data is passed to ‘saveUserData’, and finally, we log that all operations are completed.

If any of the promises are rejected (i.e., an error occurs), the catch block at the end will catch the error, allowing you to handle it appropriately. Promise chaining helps maintain a clear and organized flow of asynchronous tasks and improves the readability of asynchronous code.