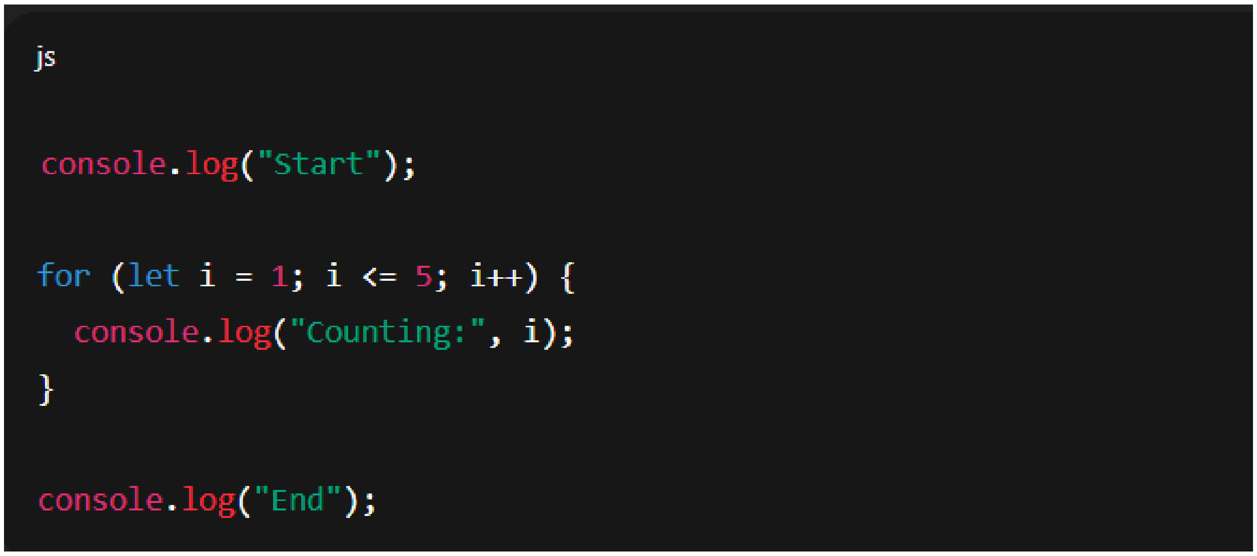
Exercise 1

Objective: Understand how JavaScript runs code line by line (blocking).

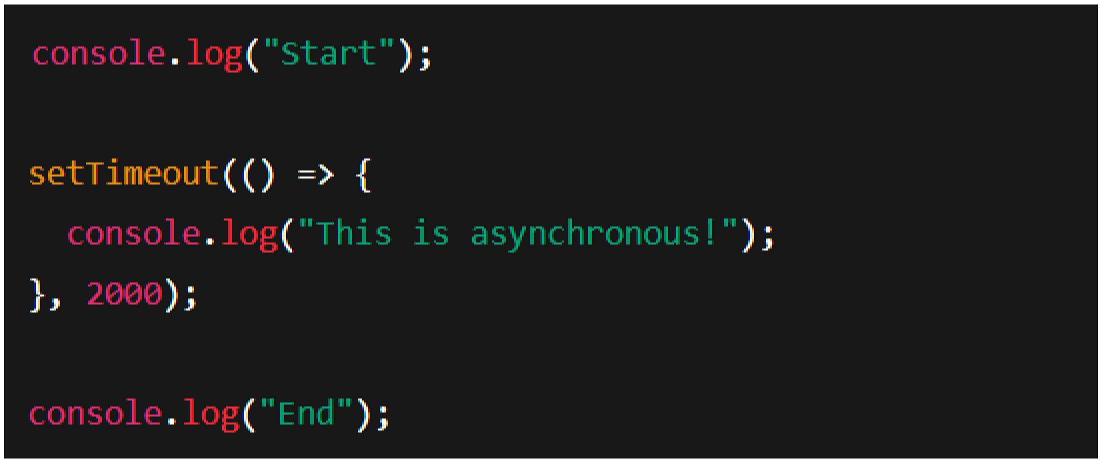


```js
console.log("Start");

for (let i = 1; i <= 5; i++) {
  console.log("Counting:", i);
}

console.log("End");
```

- Predict what will be printed and in what order.
- Then, run the code and compare the result.
- Discuss why the loop blocks other code.

Exercise 2

Objective: See how JavaScript handles asynchronous operations.



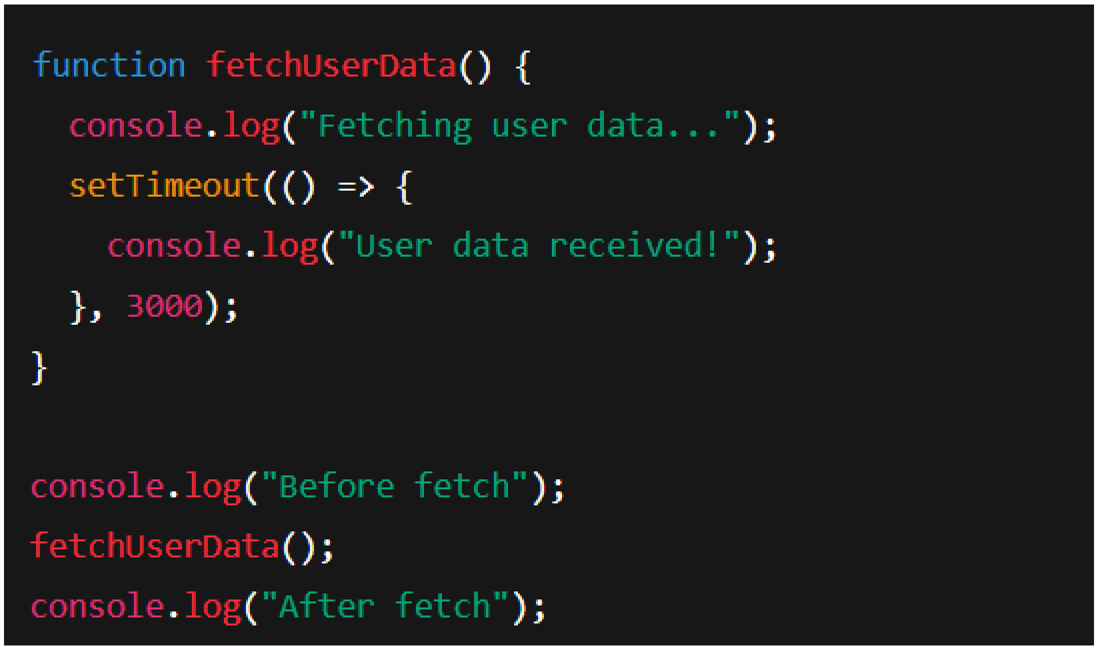
```javascript
console.log("Start");

setTimeout(() => {
  console.log("This is asynchronous!");
}, 2000);

console.log("End");
```

Questions:

- Which line prints first and why?
- What’s happening during the 2 seconds?

Exercise 3

Objective: Simulate fetching data with a delay.



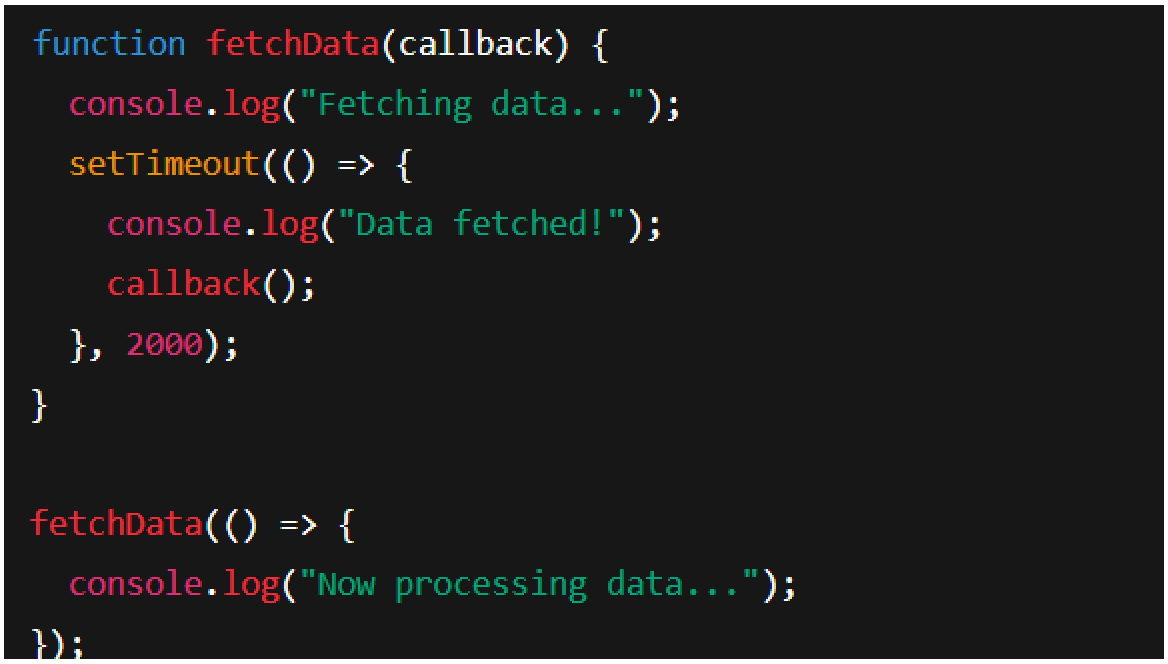
```javascript
function fetchUserData() {
  console.log("Fetching user data...");
  setTimeout(() => {
    console.log("User data received!");
  }, 3000);
}

console.log("Before fetch");
fetchUserData();
console.log("After fetch");
```

Discussion Points:

- What prints first and why?
- How can we know when the data has actually arrived?

Exercise 4

Objective: Show how callbacks are used to handle async results.



```
function fetchData(callback) {
  console.log("Fetching data...");
  setTimeout(() => {
    console.log("Data fetched!");
    callback();
  }, 2000);
}

fetchData(() => {
  console.log("Now processing data...");
});
```

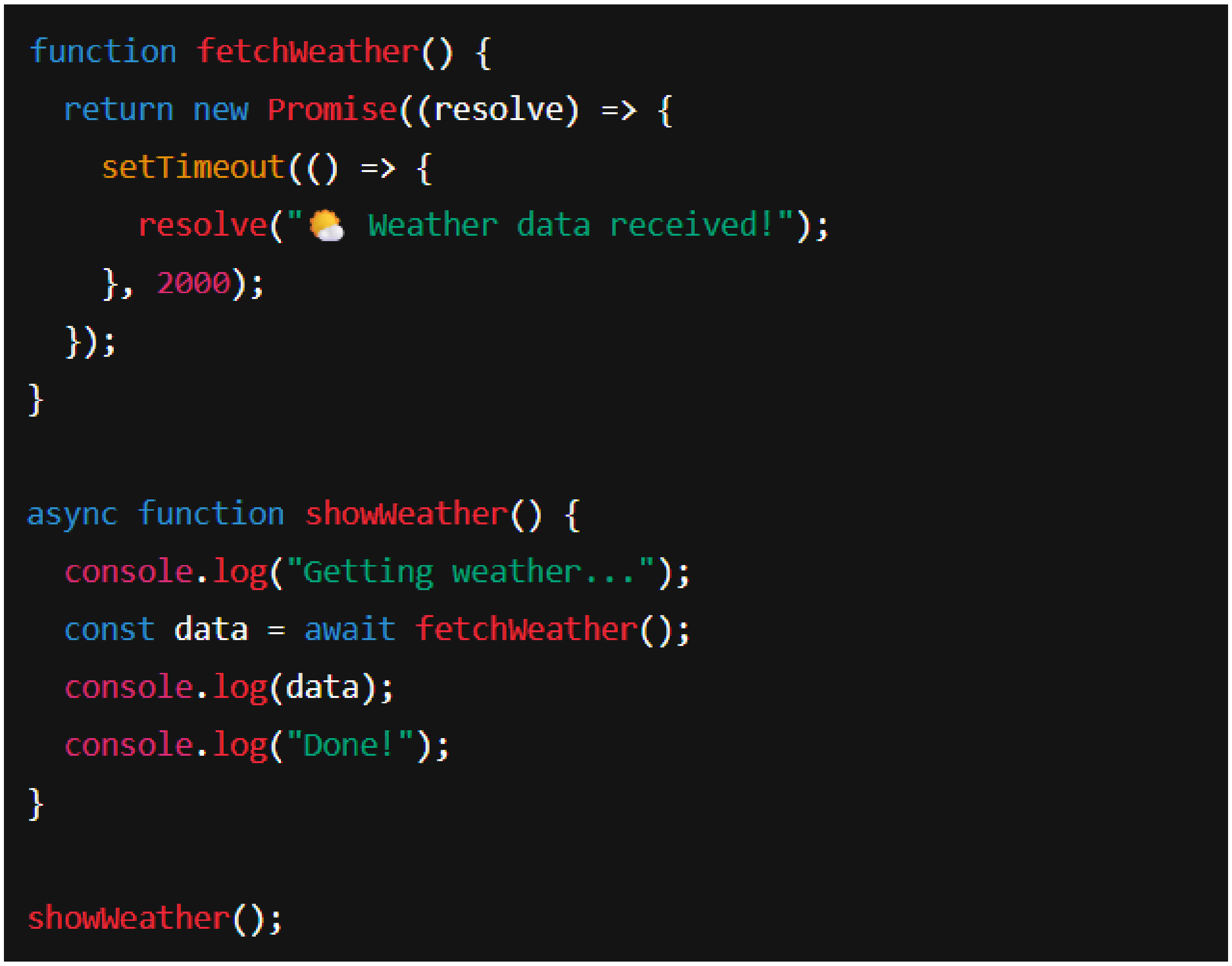
Exercise 5

Objective: Convert a callback to a Promise-based approach.

```javascript
function fetchData() {
  return new Promise((resolve, reject) => {
    console.log("Fetching data...");
    setTimeout(() => {
      const success = true;
      if (success) resolve("Data fetched successfully!");
      else reject("Error fetching data!");
    }, 2000);
  });
}

fetchData()
  .then((message) => console.log(message))
  .catch((error) => console.log(error))
  .finally(() => console.log("Operation finished"));
```

Exercise 6

Objective: Introduce async/await syntax for cleaner code.



```javascript
function fetchWeather() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("⛅ Weather data received!");
    }, 2000);
  });
}

async function showWeather() {
  console.log("Getting weather...");
  const data = await fetchWeather();
  console.log(data);
  console.log("Done!");
}

showWeather();
```

Simulated API Calls

Task:
Create three async functions:

function fetchUser() { ... }   // 2s delay
function fetchPosts() { ... } // 3s delay
function fetchComments() { ... } // 1s delay

To do::

1. Run them **one by one** using `await` (sequential).

2. Run them **at the same time** using `Promise.all()` (parallel).

Question: Which one is faster and why?