JavaScript Async/Await CheatSheet

Chetan Mahajan
@_chetanmahajan

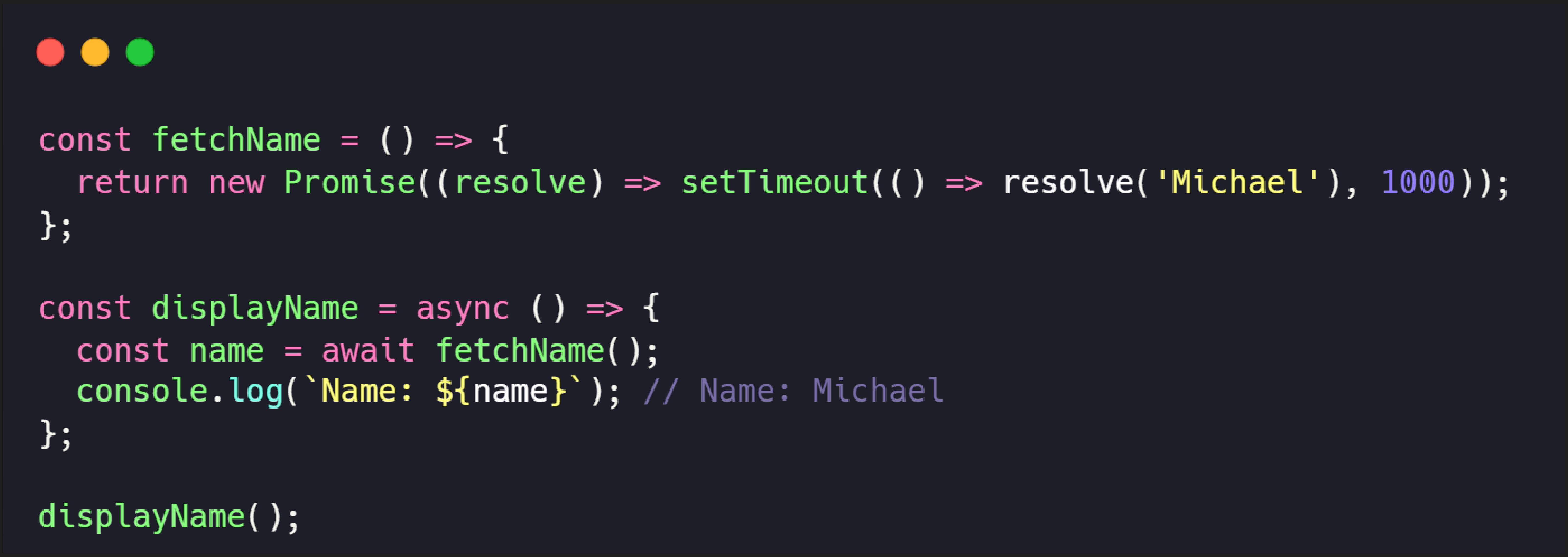
1. Basic Async Function

```
const greet = async () => {
  return 'Hello, Sarah!';
};

greet().then((message) => console.log(message)); // Hello, Sarah!
```

Use the async keyword to define a function that always returns a Promise.

Chetan Mahajan
@_chetanmahajan

2. Awaiting a Promise

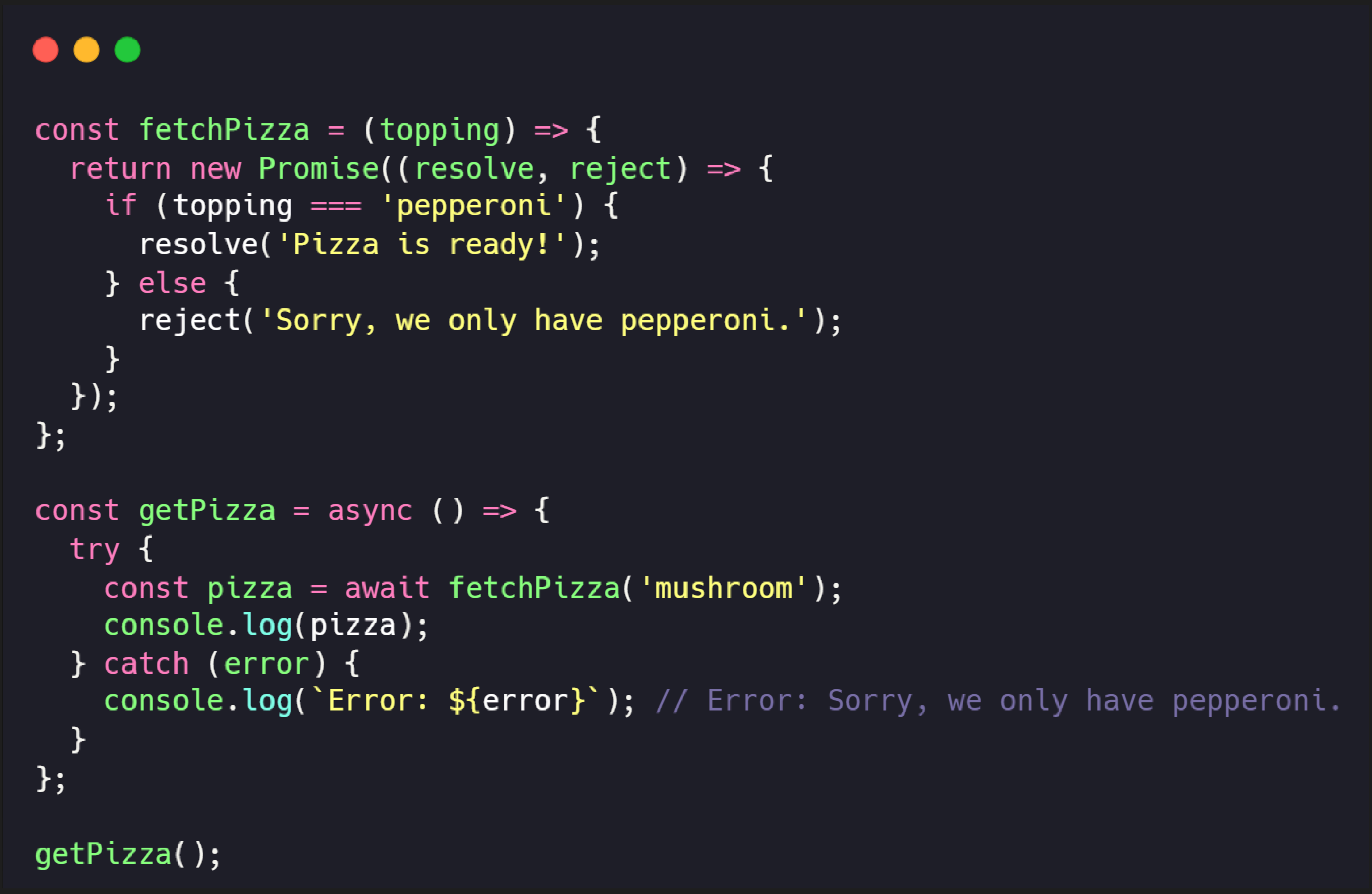


```javascript
const fetchName = () => {
  return new Promise((resolve) => setTimeout(() => resolve('Michael'), 1000));
};

const displayName = async () => {
  const name = await fetchName();
  console.log(`Name: ${name}`); // Name: Michael
};

displayName();
```

Fetching data (e.g., user's name) after waiting for a task to complete.

Chetan Mahajan
@_chetanmahajan

3. Handling Errors with Try/Catch

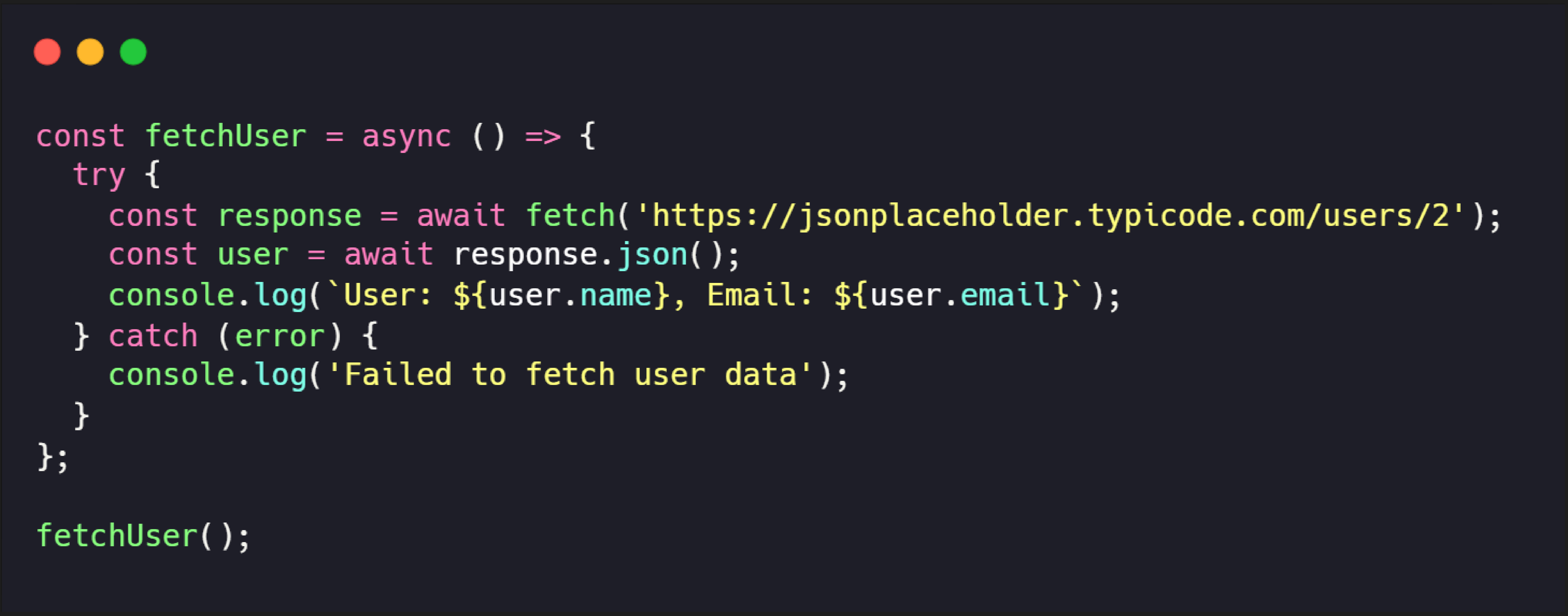


```
const fetchPizza = (topping) => {
  return new Promise((resolve, reject) => {
    if (topping === 'pepperoni') {
      resolve('Pizza is ready!');
    } else {
      reject('Sorry, we only have pepperoni.');
    }
  });
};

const getPizza = async () => {
  try {
    const pizza = await fetchPizza('mushroom');
    console.log(pizza);
  } catch (error) {
    console.log(`Error: ${error}`); // Error: Sorry, we only have pepperoni.
  }
};

getPizza();
```

Handling invalid orders when a pizza shop doesn’t have certain toppings.

Chetan Mahajan
@_chetanmahajan

4. Async/Await with API Calls



```javascript
const fetchUser = async () => {
  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/users/2');
    const user = await response.json();
    console.log(`User: ${user.name}, Email: ${user.email}`);
  } catch (error) {
    console.log('Failed to fetch user data');
  }
};

fetchUser();
```

Fetching and displaying user details from an API.

Chetan Mahajan
@_chetanmahajan

5. Multiple Awaits

```javascript
const fetchData = (data) => {
  return new Promise((resolve) => setTimeout(() => resolve(data), 1000));
};

const fetchDetails = async () => {
  const user = await fetchData('Emily');
  console.log(`User: ${user}`); // User: Emily
  const balance = await fetchData(300);
  console.log(`Balance: $${balance}`); // Balance: $300
};

fetchDetails();
```

Handle sequential tasks where each step depends on the previous one.

Chetan Mahajan
@_chetanmahajan

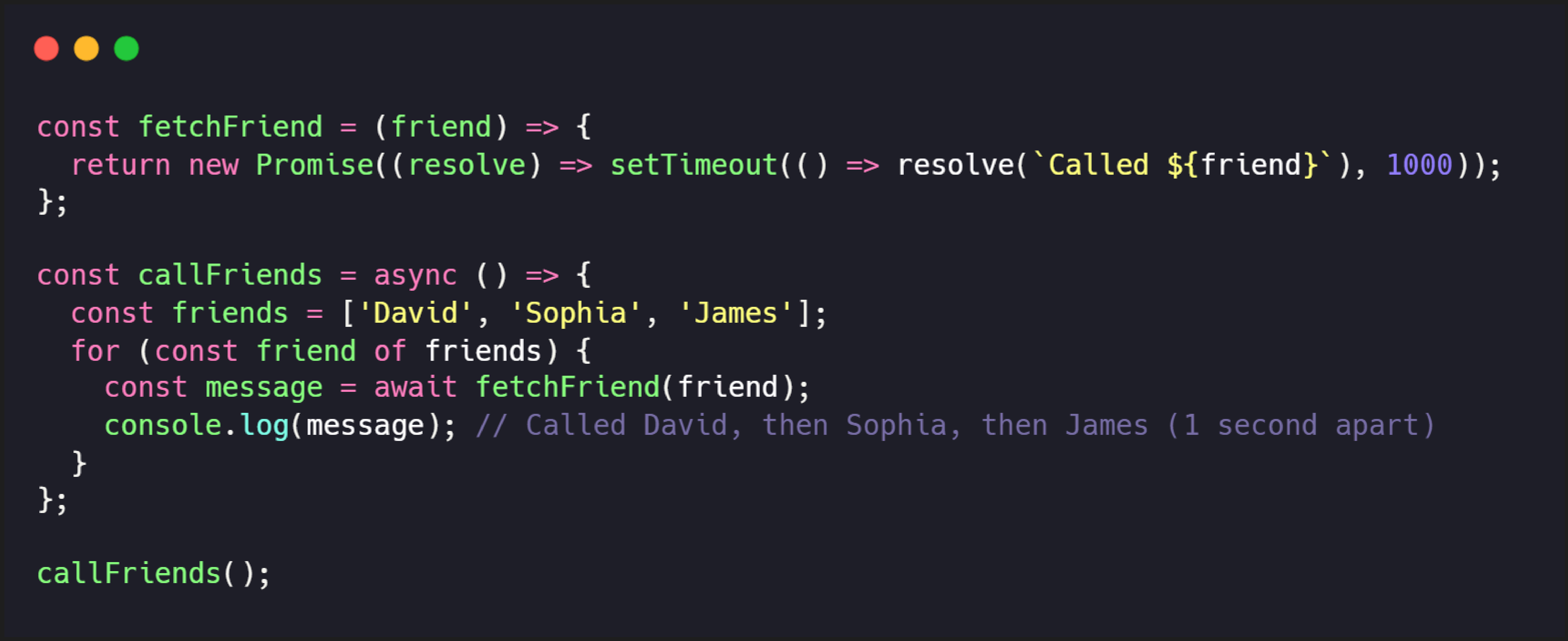
6. Parallel Execution with Promise.all

```
const fetchMovies = () => new Promise((resolve) =>
setTimeout(() => resolve('Movies fetched'), 2000));
const fetchShows = () => new Promise((resolve) =>
setTimeout(() => resolve('Shows fetched'), 1000));

const getEntertainment = async () => {
  const [movies, shows] = await Promise.all([fetchMovies(),
fetchShows()]);
  console.log(`${movies} & ${shows}`); // Movies fetched &
Shows fetched
};

getEntertainment();
```

Fetching movies and TV shows at the same time to save time.

Chetan Mahajan
@_chetanmahajan

7. Using Async Functions Inside Loops



```javascript
const fetchFriend = (friend) => {
  return new Promise((resolve) => setTimeout(() => resolve(`Called ${friend}`), 1000));
};

const callFriends = async () => {
  const friends = ['David', 'Sophia', 'James'];
  for (const friend of friends) {
    const message = await fetchFriend(friend);
    console.log(message); // Called David, then Sophia, then James (1 second apart)
  }
};

callFriends();
```

Sequentially calling friends (e.g., updating contact list).

Chetan Mahajan
@_chetanmahajan

8. Combining Async/Await with Conditions

```javascript
const fetchDiscount = (isMember) => {
  return new Promise((resolve, reject) => {
    if (isMember) resolve('You got a 10% discount!');
    else reject('No discount available.');
  });
};

const checkDiscount = async (name, isMember) => {
  try {
    const discount = await fetchDiscount(isMember);
    console.log(`${name}, ${discount}`);
  } catch (error) {
    console.log(`${name}, ${error}`);
  }
};

checkDiscount('Liam', true);  // Liam, You got a 10% discount!
checkDiscount('Emma', false); // Emma, No discount available.
```

Checking membership discounts for users.

Chetan Mahajan
@_chetanmahajan

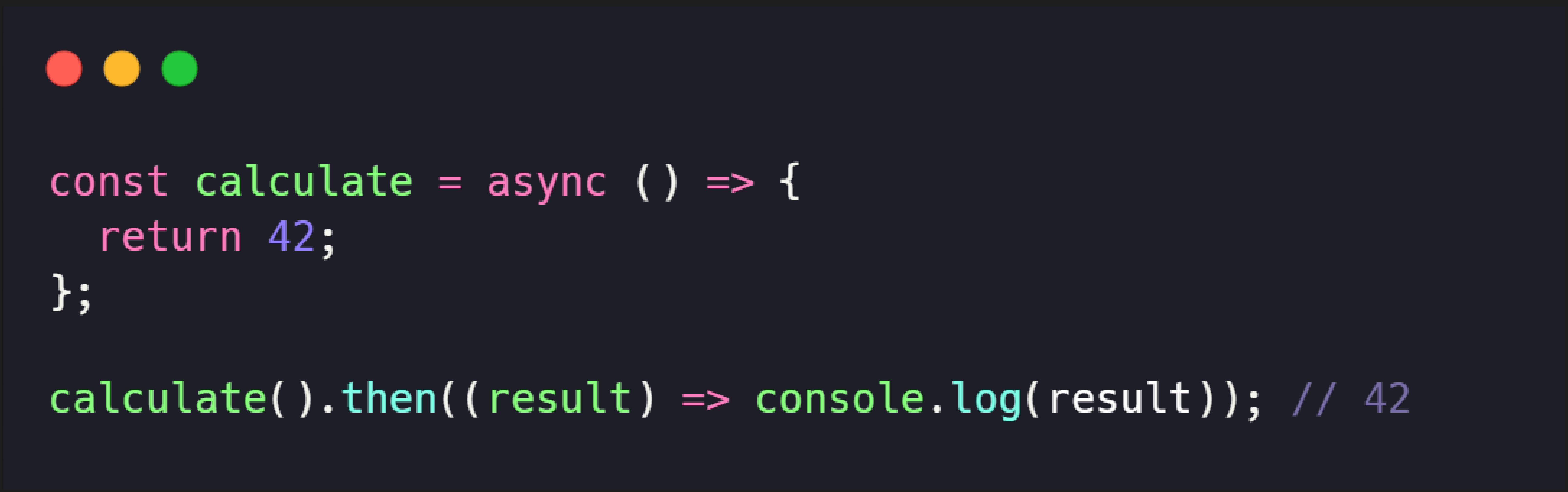
9. Using Async/Await in DOM Events

```javascript
document.getElementById('loadBtn').addEventListener('click', async () => {
  const fetchMessage = () => new Promise((resolve) => setTimeout(() =>
resolve('Data loaded!'), 1500));
  const message = await fetchMessage();
  console.log(message); // Data loaded!
});
```

Displaying feedback when a button is clicked (e.g., "Loading...").

Chetan Mahajan
@_chetanmahajan

10. Returning Values from Async Functions



```
const calculate = async () => {
  return 42;
};

calculate().then((result) => console.log(result)); // 42
```

Async functions always return a Promise, so you can use .then() to access the result.

Chetan Mahajan
@_chetanmahajan

Are you looking for Front-end Developer Job?

If yes, Check the link in bio to get Interview Kit and Start Preparing for your Next Interview