JavaScript Promises, Async and Await Practice Exercises

Exercise 1: Basic Promise

Create a function `getNumber` that returns a Promise. The Promise should resolve with a number after 2 seconds.

Instructions:

1. Create a function `getNumber` that returns a Promise.
2. Inside the Promise, use `setTimeout` to resolve the number `5` after 2 seconds.
3. Handle the resolution of the Promise using `.then()`.

Exercise 2: Chaining Promises

Create two functions `getFirstName` and `getLastName`, which return promises. Chain them so that you get the full name by combining both first and last names.

Instructions:

1. `getFirstName` returns a Promise that resolves with 'Ravi'.
2. `getLastName` returns a Promise that resolves with 'Sharma'.
3. Chain both promises and log the full name (e.g., 'Ravi Sharma').

Exercise 3: Handling Errors with Promises

Create a function `getData` that returns a Promise. If the data is fetched successfully, it should resolve with 'Data fetched', otherwise, it should reject with an error message.

Instructions:

1. `getData` should randomly either resolve or reject after 2 seconds (use `Math.random()` for random decision).

2. Handle both the resolved and rejected cases using `.then()` and `.catch()`.

Exercise 4: Async and Await Syntax

Convert the following code to use `async` and `await` instead of `.then()` and `.catch()`.

```js
function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('Data received');
    }, 1000);
  });
}

fetchData()
  .then((data) => {
    console.log(data);
  })
  .catch((err) => {
    console.log(err);
  });
```

Instructions:

1. Use `async` to define a function that handles the asynchronous operation.

2. Replace `.then()` and `.catch()` with `await` and `try/catch`.

Exercise 5: Multiple Async/Await

Write an async function that simulates fetching data from two different APIs. One function returns user data, and another returns post data. Both are simulated with timeouts.

Instructions:

1. Create two async functions, `fetchUserData` (which resolves with `{ name: 'Aarav Kumar', age: 21 }`) and `fetchPostData` (which resolves with `{ title: 'JavaScript Basics' }`).
2. In the main async function, fetch both user data and post data in parallel using `Promise.all()`.
3. Use `await` to get the results and log them.

Exercise 6: Async/Await with Timeout

Write an async function that resolves after 3 seconds. If it takes more than 3 seconds, it should reject with a timeout error.

Instructions:

1. Create a function `fetchDataWithTimeout` that simulates fetching data.
2. Use `setTimeout` to make the function reject after 3 seconds if the operation takes too long.

Exercise 7: Sequential Async/Await Execution

Write an async function that fetches data from three different sources one after another, simulating 3 different API calls (use `setTimeout`).

Instructions:

1. Use `await` to ensure each API call completes before starting the next one.
2. Log the result of each API call after it completes.
   - First API: Returns `{ name: 'Priya Gupta', city: 'Mumbai' }`
   - Second API: Returns `{ job: 'Software Engineer', company: 'TCS' }`
   - Third API: Returns `{ hobbies: ['Reading', 'Coding'] }`

Exercise 8: Error Handling with Async/Await

Modify the previous exercise to handle errors. If any API call fails, reject the promise and catch the error.

Instructions:

1. Use `try/catch` inside the async function to catch any potential errors.

2. Log the error message if one of the API calls fails.