Testing Exercises

1. Fundamentals: String Utilities

Objective: Write unit tests for string utility functions.

Code to Test:

```
function capitalize(word) {
    if (!word) return "";
    return word[0].toUpperCase() + word.slice(1);
}

function reverseString(str) {
    return str.split("").reverse().join("");
}
```

Exercise:

- Write tests to validate the capitalize function, including handling empty strings and single-character words.
- Write tests for reverseString, including edge cases with palindromes and empty strings.

---

2. Error Handling: Array Index

Objective: Test a function that accesses an array by index and handles out-of-bounds cases.

Code to Test:

```
function getElement(arr, index) {
    if (index < 0 || index >= arr.length) {
        throw new Error("Index out of bounds");
    }
    return arr[index];
}
```

Exercise:

- Write tests for valid index values.
- Write tests to check if the error is thrown for negative indices and out-of-range indices.

---

3. Async Functions: Delayed Greeting

Objective: Test an asynchronous function with a delay.

Code to Test:

```
function delayedGreeting(name, delay) {
    return new Promise((resolve) => {
        setTimeout(() => {
            resolve(`Hello, ${name}!`);
        }, delay);
    });
}
```

Exercise:

- Write tests for the resolved greeting message.
- Use a mock timer to validate that the function respects the delay.

---

4. Mocking: Notification Service

Objective: Test a notification function using mocks.

Code to Test:

```
function sendNotification(notificationService, message) {
    const status = notificationService.send(message);
    return status ? "Notification Sent" : "Failed to Send";
}
```

Exercise:

- Mock notificationService to simulate both successful and failed notification sending.
- Write tests to ensure the return message matches the scenario.

---

5. Spying: DOM Manipulation

Objective: Test a DOM manipulation function using spies.

Code to Test:

```
function toggleVisibility(element) {
    if (element.style.display === "none") {
        element.style.display = "block";
    } else {
        element.style.display = "none";
    }
}
```

Exercise:

- Use a spy to check if the style.display property changes correctly.
- Write tests to validate toggling visibility when the element is initially visible or hidden.

Bonus Challenge: Integrate All Concepts

Objective: Create a function that fetches user data, validates it, and displays it in the DOM.

Code to Test:

```
async function fetchAndDisplayUser(apiService, userId,
element) {
    try {
        const user = await apiService.getUser(userId);
        if (!user.name) throw new Error("Invalid user data");
        element.textContent = `Hello, ${user.name}`;
    } catch (error) {
        element.textContent = error.message;
    }
```

```
}
```

Exercise:

- Mock the apiService to test successful and failed user fetch scenarios.
- Spy on the DOM element's textContent property to validate correct content updates.