Q1. Write a JavaScript function called outerFunction that takes a parameter and returns an inner function. The inner function should access both the parameter of outerFunction and a variable declared within outerFunction. Demonstrate how lexical scoping allows the inner function to maintain access to these variables even after outerFunction has finished executing.

```
function outerFunction(outerParam) {
  let innerVariable = "I'm inside outerFunction!";

  return function innerFunction() {
    console.log("Outer parameter:", outerParam);
    console.log("Inner variable:", innerVariable);
  };
}

// Create an instance of the inner function
const inner = outerFunction("I'm from outerFunction!");

// Call the inner function, even after outerFunction has executed
inner();
```

Output:

```
Outer parameter: I'm from outerFunction!
Inner variable: I'm inside outerFunction!
```

Explanation:

- Lexical scoping allows the innerFunction to access both outerParam (the parameter of outerFunction) and innerVariable (a variable declared within outerFunction), even though outerFunction has finished executing when innerFunction is called.

Q2. Create a JavaScript program that demonstrates the basic usage of regular expressions. Write a function that takes a regex pattern and a string as input and returns true if there is a match, and false otherwise. Test the function with various patterns and strings.

```
function testRegex(pattern, str) {
  const regex = new RegExp(pattern);
  return regex.test(str);
}
```

```
// Test the function with various patterns and strings

console.log(testRegex('abc', 'abcdef'));   // Output: true (matches "abc")

console.log(testRegex('\\d+', '123abc'));  // Output: true (matches digits "123")

console.log(testRegex('[A-Za-z]+', 'Hello123')); // Output: true (matches "Hello")

console.log(testRegex('^abc', 'abcdef')); // Output: true (matches "abc" at the start)

console.log(testRegex('xyz', 'abcdef'));  // Output: false (no match)
```

Explanation:

- The function testRegex uses RegExp to test whether a given pattern matches a string. We test it with a variety of patterns like matching specific characters (abc), digits (\d+), and start-of-string (^abc).

Q3. Write a JavaScript program that demonstrates the use of character classes in regular expressions. Create a function that searches for specific character classes in a given string and returns the matches. Test the function with patterns for digits, uppercase letters, lowercase letters, and special characters.

```
function findCharacterClassMatches(pattern, str) {

   const regex = new RegExp(pattern, 'g');

   return str.match(regex);

}


// Test the function with various character classes

console.log(findCharacterClassMatches('\\d', 'abc123def456'));  // Output: [ '1', '2', '3', '4', '5', '6' ] (digits)

console.log(findCharacterClassMatches('[A-Z]', 'abcDEFxyz'));   // Output: [ 'D', 'E', 'F' ] (uppercase letters)

console.log(findCharacterClassMatches('[a-z]', 'abcDEFxyz'));   // Output: [ 'a', 'b', 'c', 'x', 'y', 'z' ] (lowercase letters)

console.log(findCharacterClassMatches('[^A-Za-z0-9]', 'abc@123!')); // Output: [ '@', '!' ] (special characters)
```

Explanation:

- This function demonstrates the use of character classes in regular expressions:
  - \d for digits.
  - [A-Z] for uppercase letters.

- **[a-z] for lowercase letters.**
- **[^A-Za-z0-9] for non-alphanumeric characters.**

Q4. Create a JavaScript program that takes a regex pattern and a string as input. Write a function that not only checks if there is a match but also extracts specific parts of the matched text using groups. Test the function with patterns that include groups to capture different parts of a date (e.g., day, month, and year) from a given string.

```
function extractDateParts(pattern, str) {
   const regex = new RegExp(pattern);
   const match = str.match(regex);
   if (match) {
      return {
         day: match[1],
         month: match[2],
         year: match[3]
      };
   } else {
      return null;
   }
}

// Pattern to match a date in format "DD-MM-YYYY"
const datePattern = '(\\d{2})-(\\d{2})-(\\d{4})';

// Test the function with a date string
console.log(extractDateParts(datePattern, 'The date is 15-08-2025.'));
```

Output:

```
{
  day: '15',
  month: '08',
  year: '2025'
}
```

Explanation:

- **The function uses a regular expression to capture different parts of a date:**
  - **(\d{2}) captures the day (two digits).**
  - **(\d{2}) captures the month (two digits).**
  - **(\d{4}) captures the year (four digits).**
- **The match method returns an array where the first item is the entire match, and subsequent items correspond to the captured groups.**

Q5. You are building a shipping application. Write a program that takes the type of package ("standard", "express", or "overnight") and uses a switch statement to calculate and print the estimated delivery time based on the package type. For example, "standard" might take 3-5 days, "express" 1-2 days, and "overnight" would be delivered the next day.

```
function calculateDeliveryTime(packageType) {
  let deliveryTime;

  switch (packageType.toLowerCase()) {
    case 'standard':
      deliveryTime = '3-5 days';
      break;
    case 'express':
      deliveryTime = '1-2 days';
      break;
    case 'overnight':
      deliveryTime = 'Next day delivery';
      break;
    default:
      deliveryTime = 'Invalid package type';
  }

  console.log(`Estimated delivery time for ${packageType}: ${deliveryTime}`);
}
```

```
// Test the function with different package types
calculateDeliveryTime('standard');   // Output: Estimated delivery time for standard: 3-5 days
calculateDeliveryTime('express');    // Output: Estimated delivery time for express: 1-2 days
calculateDeliveryTime('overnight');  // Output: Estimated delivery time for overnight: Next day delivery
calculateDeliveryTime('premium');    // Output: Estimated delivery time for premium: Invalid package type
```

Explanation:

- **The switch statement checks the packageType and assigns an appropriate delivery time. If an invalid package type is provided, it outputs "Invalid package type."**
