Q1. Create an arrow function called square that takes a number as an argument and returns its square.

```
const square = (num) => num * num;


// Test the square function with a number
let result = square(5);
console.log(result);  // Output: 25
```

In this example, the arrow function square takes a number and returns its square.

Q2. Create a JavaScript function called generateGreeting that takes a name as an argument and returns a personalized greeting message.

```
function generateGreeting(name) {
   return `Hello, ${name}! Welcome!`;
}


// Test the function with three different names
console.log(generateGreeting('Alice'));  // Output: Hello, Alice! Welcome!
console.log(generateGreeting('Bob'));    // Output: Hello, Bob! Welcome!
console.log(generateGreeting('Charlie'));// Output: Hello, Charlie! Welcome!
```

Here, the function generateGreeting takes a name and returns a personalized greeting message.

Q3. Create an IIFE (Immediately Invoked Function Expression) that calculates the square of a number and immediately displays the result.

```
(function(num) {
   console.log(num * num);
})(4);  // Output: 16
```

This is an IIFE (Immediately Invoked Function Expression) that calculates and displays the square of 4 immediately after being defined.

Q4. Write a JavaScript function called calculateTax that takes an income as an argument and returns the amount of tax to be paid. Use a closure to handle different tax rates based on income ranges.

```
function calculateTax(income) {
```

```
  const taxRate = (function() {
    if (income <= 50000) {
      return 0.1;  // 10% tax for income <= 50,000
    } else if (income <= 100000) {
      return 0.2;  // 20% tax for income <= 100,000
    } else {
      return 0.3;  // 30% tax for income > 100,000
    }
  })();

  return income * taxRate;
}

// Test the calculateTax function with various incomes
console.log(calculateTax(40000));  // Output: 4000 (10% of 40,000)
console.log(calculateTax(75000));  // Output: 15000 (20% of 75,000)
console.log(calculateTax(120000)); // Output: 36000 (30% of 120,000)
```

Here, the closure helps to decide the tax rate based on income, and the function returns the tax amount accordingly.

Q5. Write a JavaScript function called factorial that calculates the factorial of a non-negative integer using recursion.

```
function factorial(n) {
  if (n === 0 || n === 1) {
    return 1;
  }
  return n * factorial(n - 1);
}

// Test the factorial function with different inputs
console.log(factorial(5)); // Output: 120 (5! = 5 * 4 * 3 * 2 * 1)
console.log(factorial(0)); // Output: 1 (0! = 1)
```

console.log(factorial(3)); // Output: 6 (3! = 3 * 2 * 1)

This function calculates the factorial of a number recursively. The base case is when n is 0 or 1, returning 1.

Q6. Write a JavaScript function called curry that takes a function as an argument and returns a curried version of that function. The curried function should accept arguments one at a time and return a new function until all arguments are provided. Then, it should execute the original function with all arguments.

```
function curry(fn) {
  return function curried(...args) {
    if (args.length >= fn.length) {
      return fn(...args);
    } else {
      return function(...nextArgs) {
        return curried(...args, ...nextArgs);
      };
    }
  };
}

// Example: Function that adds two numbers
function add(a, b) {
  return a + b;
}

// Curry the add function
const curriedAdd = curry(add);

// Test the curried function
console.log(curriedAdd(5)(10));  // Output: 15
console.log(curriedAdd(3)(7));   // Output: 10
```

In this example, the curry function transforms a function like add into a curried version that accepts arguments one by one. When all arguments are provided, the original function is executed.