CODE EXERCISE

World of JavaScript Closures

CODING EXERCISES TEST YOUR SKILLS

Introduction 🚀 Launching into the World of JavaScript Closures! 🚀

Series of 10 meticulously designed JavaScript exercises, exclusively focusing on the concept of closures - a core aspect of JavaScript that often perplexes even seasoned developers. 🤓💻

Each exercise is thoughtfully crafted to not only challenge but also to enlighten you about the different facets of closures. They range from basic implementations to more complex scenarios, ensuring a comprehensive understanding of the topic. 🌟

Here's a sneak peek at what's included:

- Basic Closure Mechanics
- Implementing Counter Functions
- Argument Remembering Functions
- Creating Private Variables
- Function Factories
- Caching with Closures (Memoization)
- Data Encapsulation Techniques
- Handling Loops and Closures
- Event Listeners using Closures
- Currying Functions

Whether you're a budding programmer eager to crack the mystery of closures or a pro developer looking to brush up your skills, these exercises are tailored for you. They are perfect for interview prep, coding practice, or even as a quick refresher! 📘👨‍💻👩‍💻

Exercise 1: Basic Closure

Problem Statement:

Create a function createGreeting that takes a name and returns another function. The returned function should return a greeting string when called.

Hint/Explanation:

A closure is a function that remembers the variables from the place where it is defined, regardless of where it is executed later.

Solution:

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

const greetAlice = createGreeting("Alice");
console.log(greetAlice()); // "Hello, Alice!"
```

Exercise 2: Counter Function

Problem Statement:

Create a function createCounter that returns a function. When the returned function is called, it should return an incremented value starting from 1.

Hint/Explanation:

The inner function can access and modify variables defined in the outer function.

Solution:

```
function createCounter() {
   let count = 0;
   return function() {
      count += 1;
      return count;
   };
}

const counter = createCounter();
console.log(counter()); // 1
console.log(counter()); // 2
```

Exercise 3: Remembering Arguments

Problem Statement:

Create a function rememberArgs that takes two arguments and returns a function. When the returned function is called, it should return the sum of the arguments passed to rememberArgs.

Hint/Explanation:

Closures keep track of the variables from their containing scope.

Solution:

```
function rememberArgs(a, b) {
    return function() {
        return a + b;
    };
}

const adder = rememberArgs(2, 3);
console.log(adder()); // 5
```

Exercise 4: Private Variables

Problem Statement:

Create a function createBankAccount that initializes a balance (a private variable) and returns an object with two methods deposit and withdraw, manipulating the balance.

Hint/Explanation:

Closures allow for private variables that cannot be accessed directly from outside the function.

Solution:

```
function createBankAccount(initialBalance) {
    let balance = initialBalance;
    return {
```

```
    deposit(amount) {
        balance += amount;
        return balance;
    },
    withdraw(amount) {
        if (amount > balance) {
            return "Insufficient funds";
        }
        balance -= amount;
        return balance;
    }
  };
}

const account = createBankAccount(100);
console.log(account.deposit(50)); // 150
console.log(account.withdraw(70)); // 80
```

Exercise 5: Function Factories

Problem Statement:

Create a function multiplier that takes a number x and returns a new function. The returned function should take a number y and return the product of x and y.

Hint/Explanation:

This pattern, where a function is used to create new functions, is a powerful use of closures.

Solution:

```
function multiplier(x) {
   return function(y) {
      return x * y;
   };
}

const double = multiplier(2);
console.log(double(5)); // 10
```

Exercise 6: Caching/Memoization

Problem Statement:

Implement a simple caching mechanism for a function that calculates the factorial of a number. Use closures to remember previously calculated results.

Hint/Explanation:

Closures can be used to implement memoization, an optimization technique to speed up function calls by caching results.

Solution:

```
function factorial() {
```

```
    const cache = {};
    return function innerFact(n) {
        if (n in cache) {
            return cache[n];
        }
        if (n === 0 || n === 1) {
            return 1;
        }
        const result = n * innerFact(n - 1);
        cache[n] = result;
        return result;
    };
}

const fact = factorial();
console.log(fact(5)); // 120
console.log(fact(6)); // 720, faster due to caching
```

Exercise 7: Encapsulation with Closures

Problem Statement:

Create a function createPerson that takes a name and age and returns an object with methods to get and set the name and age. The name and age should not be directly accessible.

Hint/Explanation:

This exercise demonstrates how closures can be used to encapsulate and protect data.

Solution:

```
function createPerson(name, age) {
   let privateName = name;
   let privateAge = age;

   return {
      getName() {
         return privateName;
      },
      setName(newName) {
         privateName = newName;
      },
      getAge() {
         return privateAge;
      },
      setAge(newAge) {
         privateAge = newAge;
      }
   };
}
```

```
const person = createPerson("Alice", 25);
console.log(person.getName()); // Alice
person.setName("Bob");
console.log(person.getName()); // Bob
```

Exercise 8: Loop and Closures

Problem Statement:

Create a function createFunctions that returns an array of functions. Each function, when called, should return its index in the array.

Hint/Explanation:

This exercise is tricky due to how closures and loops interact, especially in regards to variable scoping.

Solution:

```
function createFunctions(n) {
   const functions = [];

   for (let i = 0; i < n; i++) {
      functions.push((function(index) {
         return function() {
            return index;
         };
```

```
        })(i));
    }

    return functions;
}

const funcs = createFunctions(3);
console.log(funcs[0]()); // 0
console.log(funcs[1]()); // 1
console.log(funcs[2]()); // 2
```

Exercise 9: Event Listeners and Closures

Problem Statement:

Create a function setupButtons that sets up event listeners on buttons. Each button, when clicked, should alert its position in a list.

Hint/Explanation:

This exercise shows how closures can be useful in event handling, allowing access to the loop index in event listeners.

Solution:

```
function setupButtons() {
    const buttons = document.querySelectorAll("button");
```

```
    for (let i = 0; i < buttons.length; i++) {
        buttons[i].addEventListener("click", function() {
            alert(`Button ${i + 1} clicked`);
        });
    }
}

// Call setupButtons() after the DOM has loaded
```

Exercise 10: Currying with Closures

Problem Statement:

Implement a curry function that takes a binary function and an argument, and returns a new function that can take a second argument.

Hint/Explanation:

Currying is the technique of converting a function that takes multiple arguments into a sequence of functions that each take a single argument.

Solution:

```
function curry(binaryFunc, firstArg) {
    return function(secondArg) {
        return binaryFunc(firstArg, secondArg);
    };
}
```

```
const add = (a, b) => a + b;
const addFive = curry(add, 5);
console.log(addFive(3)); // 8
```

These exercises cover various aspects and applications of closures in JavaScript, providing a mix of practical and conceptual learning opportunities. They can be adapted for different learning levels and are ideal for both instructional and self-paced learning environments.

Don't hesitate to reach out if you have questions or need more insights into any of these exercises. Let's embark on this learning journey together and unravel the potential of JavaScript closures! 💡🌐

#JavaScript #Closures #WebDevelopment #CodingChallenges #Programming #LearningToCode #JavaScriptDeveloper #FrontEndDevelopment #SoftwareEngineering #CodingIsFun #TechCommunity #CodeNewbies #100DaysOfCode #DevCommunity #LearnJavaScript #JavaScriptTips