**JavaScript Functions Assignment**

**Objective:**

This assignment will help you practice different types of functions in JavaScript including arrow functions, callbacks, higher-order functions, and function expressions.

**Part 1: Arrow Functions**

1. **Basic Arrow Function**
   * Write an arrow function that takes two arguments a and b, and returns their sum.
2. **Arrow Function with No Parameters**
   * Write an arrow function that logs a greeting message like "Hello, world!" to the console.
3. **Arrow Function with Implicit Return**
   * Write an arrow function that takes an array of numbers and returns the sum of the numbers (use the reduce() method with an arrow function).

**Part 2: Callback Functions**

1. **Callback in Array Methods**
   * Write a function “calculate” that takes an array and a callback function. The callback function will be applied to each element of the array. Use this function to:
     + Find the square of each element in the array.
     + Find the cube of each element in the array.
2. **Using setTimeout() with a Callback**
   * Write a function that uses setTimeout() to simulate a delay of 2 seconds, after which it logs "Process Complete" to the console. Use a callback function to display a message before the process begins.

**Part 3: Higher-Order Functions**

1. **Function Returning a Function**
   * Write a higher-order function called createMultiplier that takes one number as an argument (multiplier) and returns another function. The returned function should take a number and multiply it by the multiplier.
     + Example: let multiplyBy3 = createMultiplier(3); multiplyBy3(5); should return 15.
2. **Higher-Order Function with Array**
   * Write a higher-order function called applyOperation that takes an array and another function (callback). The callback will determine the operation to be applied to each element (e.g., doubling, squaring, etc.). Test it with different operations like doubling, squaring, etc.
3. **map() and filter() with Arrow Functions**
   * Write an arrow function that squares each element of an array using the map() method.
   * Write an arrow function that filters out odd numbers from an array using the filter() method.

**Part 4: Function Expressions**

1. **Anonymous Function Expression**
   * Write an anonymous function that takes a string as input and returns the reverse of that string. Assign the function to a variable.
2. **Named Function Expression**
   * Write a named function expression that calculates the factorial of a given number.

**Part 5: Closures and Scopes**

1. **Closure Example**
   * Create a function counter() that contains a variable count. The function should return another function that increases the value of count by 1 and logs it to the console every time it is called.
2. **Understanding Scope**
   * Write a function that demonstrates the concept of local, block, and global scopes by declaring variables at different levels (inside a function, inside an if block, and globally).

**Part 6: this Keyword and Function Context**

1. **Arrow Function vs Regular Function**
   * Write an object person with a name property and a sayName() method. Inside sayName(), use a regular function to log the person's name to the console. Then replace the regular function with an arrow function and explain why the value of this changes.
2. **this with Callback Function**
   * Write an object with a method that accepts a callback function. Pass the method as a callback to another function and explain how the value of this is affected inside the callback.

**Part 7: Miscellaneous**

1. **IIFE (Immediately Invoked Function Expression)**
   * Write an IIFE that immediately logs "Executed Immediately!" to the console.
2. **Currying Function**
   * Write a curried function that takes three arguments, one at a time, and returns the sum of all three.

**Medium and Hard Difficulty JavaScript Functions Questions**

**Part 1: Medium Difficulty**

1. **Arrow Functions with Closures**  
   Write an arrow function createCounter that returns an object containing two methods: increment() and decrement(). These methods should modify and log a private count variable that is accessible only within the createCounter function. Demonstrate how closures work with arrow functions using this example.
2. **Higher-Order Function with Multiple Callbacks**  
   Write a higher-order function processData that takes an array of numbers and two callback functions: filterCondition and transformOperation. The filterCondition will filter elements from the array, and the transformOperation will apply a transformation (like doubling, squaring, etc.) to the filtered elements.  
   Example:  
   processData([1, 2, 3, 4, 5], filterEvenNumbers, squareElements);
3. **Currying with Arrow Functions**  
   Write a curried arrow function multiply that takes three numbers one at a time and returns the product.  
   Example:  
   multiply(2)(3)(4); should return 24.
4. **Function to Find Unique Elements**  
   Write a function findUniqueElements that takes an array of numbers and returns a new array with only the unique elements. Use a combination of filter() and higher-order functions.
5. **Debouncing with Function Expressions**  
   Implement a debounced function in JavaScript using function expressions. The debounced function should only execute the passed-in callback after it stops being called for a specific time interval. Example:  
   const debounce = (callback, delay) => {...};

**Part 2: Hard Difficulty**

1. **Memoization using Higher-Order Functions**  
   Implement a memoized version of a function that computes the nth Fibonacci number. Your function should use a closure to store previously computed values in order to make future calls more efficient.

Example:

js

Copy code

const memoizedFibonacci = (n) => {...};

1. **Function to Partially Apply Arguments (Partial Application)**  
   Write a function partial that takes a function and some arguments, and returns a new function that can take additional arguments. The goal is to apply arguments partially over multiple calls.

Example:

js

Copy code

function add(x, y, z) {

return x + y + z;

}

let partialAdd = partial(add, 2);

console.log(partialAdd(3, 5)); // Output: 10

1. **Recursive Function with Accumulator**  
   Write a recursive function that flattens a deeply nested array of arbitrary depth into a single array. You are not allowed to use array methods like flat() or reduce().

Example:

js

Copy code

let nestedArray = [1, [2, [3, [4, 5]]]];

flatten(nestedArray); // Output: [1, 2, 3, 4, 5]

1. **Throttle Function Implementation**  
   Write a throttle function that limits the execution of a given callback to only once in a specific interval, regardless of how often it’s invoked.

Example:

js

Copy code

const throttle = (callback, delay) => {...};

1. **Hard Challenge: Implement bind() from Scratch**  
   Implement a version of the bind() function in JavaScript from scratch. Your implementation should mimic the native behavior of Function.prototype.bind() and support partial application of arguments.

Example:

js

Copy code

function greet(greeting, punctuation) {

return greeting + ', ' + this.user + punctuation;

}

const person = { user: 'John' };

const boundGreet = myBind(greet, person);

console.log(boundGreet('Hello', '!')); // Output: 'Hello, John!'

**Bonus: Extra Hard Questions**

1. **Composing Functions**  
   Write a function compose that takes an arbitrary number of functions as arguments and returns a function that will run each of the functions in sequence, passing the result of each function as an argument to the next.

Example:

js

Copy code

function addTwo(x) { return x + 2; }

function multiplyByThree(x) { return x \* 3; }

const composed = compose(addTwo, multiplyByThree);

console.log(composed(5)); // Output: 21

1. **Asynchronous Control Flow with Callbacks (Waterfall Pattern)**  
   Write a waterfall function that takes an array of asynchronous functions and a final callback. Each asynchronous function should be executed in sequence, passing the result of the previous function to the next one.

Example:

js

Copy code

waterfall([fn1, fn2, fn3], finalCallback);

**W/O Advanced Topics**

**Medium Difficulty:**

1. **Arrow Functions** Write an arrow function that takes two numbers and returns the larger of the two.
2. **Callback Functions** Implement a function greet(name, callback) where callback logs the length of the name. Use an arrow function as the callback.
3. **Higher-Order Functions** Write a higher-order function multiplyBy(n) that returns a function which multiplies its argument by n. Use the returned function to multiply a number by 5.
4. **Anonymous Functions** Implement a function that takes an array of numbers and a callback function. The callback function should determine if the number is even or odd, and the function should log "Even" or "Odd" based on the result.
5. **Returning Functions** Create a function add(x) that returns a new function which adds x to its argument. Test it by passing different values to the outer and inner functions.
6. **Default Parameters** Write a function that calculates the power of a number. The second argument should have a default value of 2, making it a square function by default.

**Hard Difficulty:**

1. **Recursion with Arrow Functions** Implement a recursive arrow function that calculates the factorial of a given number.
2. **Closures** Write a closure that maintains a private count and provides two functions: one to increment the count and one to retrieve the current count.
3. **Higher-Order Functions with Arrays** Using map(), write a function that takes an array of numbers and returns an array of the squares of those numbers. Implement the squaring logic using an arrow function inside map().
4. **Function Composition** Write two functions add5 and double. Use a higher-order function to compose these two functions such that the result is first doubled and then increased by 5.
5. **Callback Hell** Write a series of nested functions (callback hell style) where each function prints a message after a delay. Use setTimeout() for the delays and ensure the functions execute sequentially.
6. **Functional Programming** Write a function chain(arr, fn1, fn2, fn3) that takes an array and three functions, applying each function to the array in sequence (functional chaining).