### **Lab 5:** **Lambda Expression in Swift**

In Swift, lambda expressions are known as closures. This lab exercise will help you practice using closures, including defining and using simple closures, closures with parameters and return values, and capturing values.

**Part 1: Basic Closures**

1. **Simple Closure:**

* Define a simple closure that prints "Hello, Swift!".

let greet = {

print("Hello, Swift!")

}

// Call the closure

greet()

1. **Closure with Parameters:**

* Define a closure that takes a String parameter and prints a greeting message with that name.

let greetUser: (String) -> Void = { name in

print("Hello, \(name)!")

}

// Call the closure

greetUser("Alice")

1. **Closure with Return Value:**

* Define a closure that takes two Int parameters and returns their sum.

let add: (Int, Int) -> Int = { a, b in

return a + b

}

// Call the closure and print the result

let sum = add(3, 5)

print("Sum: \(sum)")

**Part 2: Closures with Shorthand Syntax**

1. **Shorthand Argument Names:**

Redefine the add closure using shorthand argument names.

let addShorthand: (Int, Int) -> Int = {

return $0 + $1

}

// Call the closure and print the result

let sumShorthand = addShorthand(3, 5)

print("Sum with shorthand: \(sumShorthand)")

1. **Single Expression Closures:**

* Simplify the addShorthand closure to a single expression closure.

let addSingleExpression: (Int, Int) -> Int = { $0 + $1 }

// Call the closure and print the result

let sumSingleExpression = addSingleExpression(3, 5)

print("Sum with single expression: \(sumSingleExpression)")

**Part 3: Capturing Values**

1. **Capturing Values:**

* Define a closure that captures and modifies a variable from its surrounding context.

var counter = 0

let incrementCounter = {

counter += 1

print("Counter: \(counter)")

}

// Call the closure multiple times

incrementCounter()

incrementCounter()

incrementCounter()

**Part 4: Practical Examples**

1. **Sorting an Array:**

* Declare an array of integers called numbers and initialize it with some values.
* Use the sorted(by:) method with a closure to sort the array in descending order and print the result.

let numbers = [5, 3, 8, 2, 1, 9]

let sortedNumbers = numbers.sorted(by: { $0 > $1 })

print("Sorted numbers in descending order: \(sortedNumbers)")

1. **Filtering an Array:**

* Declare an array of integers called numbers and initialize it with some values.
* Use the filter method with a closure to filter out the even numbers and print the result.

let numbers = [5, 3, 8, 2, 1, 9]

let evenNumbers = numbers.filter { $0 % 2 == 0 }

print("Even numbers: \(evenNumbers)")

1. **Mapping an Array:**

* Declare an array of integers called numbers and initialize it with some values.
* Use the map method with a closure to create a new array with each number squared and print the result.

let numbers = [5, 3, 8, 2, 1, 9]

let squaredNumbers = numbers.map { $0 \* $0 }

print("Squared numbers: \(squaredNumbers)")

**Part 5: Higher-Order Functions**

1. **Using a Function that Accepts a Closure:**

* Write a function called applyOperation that takes two Int parameters and a closure that performs an operation on these parameters. The function should return the result of the operation.

func applyOperation(a: Int, b: Int, operation: (Int, Int) -> Int) -> Int {

return operation(a, b)

}

// Use the function with different operations

let sum = applyOperation(a: 3, b: 5, operation: { $0 + $1 })

let product = applyOperation(a: 3, b: 5, operation: { $0 \* $1 })

print("Sum: \(sum)")

print("Product: \(product)")

**Summary**

This exercise covers the basics of using closures in Swift, including defining simple closures, closures with parameters and return values, capturing values, and using closures with higher-order functions. By completing these tasks, you will become familiar with writing and using closures to perform various operations in Swift. Experiment with additional operations and modifications to further enhance your understanding of Swift closures..