Date: 19.02.2021

Kavya Casshyap
Mentor: Mr. Amit Kumar
SPOC: Mr. Abhishek Maurya

SWIFT INTERMEDIATE Assignment

Question 1.

Write a function called siftBeans(fromGroceryList:) that takes a grocery list (as an array of strings) and “sifts out” the beans from the other groceries. The function should take one argument that has a parameter name called list, and it should return a named tuple of the type (beans: [String], otherGroceries: [String]).

Hint:

Here is an example of how you should be able to call your function and what the result should be:

```
let result = siftBeans(fromGroceryList: ["green beans",
                                         "milk",
                                         "black beans",
                                         "pinto beans",
                                         "apples"])


result.beans == ["green beans", "black beans", "pinto beans"] // true

result.otherGroceries == ["milk", "apples"] // true”
```

swiftinter_1

```swift
import UIKit

func siftBeans(fromGroceryList list:[String]) ->(beans:[String], otherGroceries: [String]) {
    return (
        list.filter { $0.hasSuffix("beans")},
        list.filter { !$0.hasSuffix("beans")}
    )
}
let result = siftBeans(fromGroceryList: ["green beans",
"milk",
"black beans",
"pinto beans",
"apples"])

result.beans == ["green beans", "black beans", "pinto beans"]

result.otherGroceries == ["milk", "apples"]
```

```
(["green beans", "black beans", "pinto beans"], ["milk", "apples"])
(5 times)
(5 times)

(["green beans", "black beans", "pinto beans"], ["milk", "apples"])

true

true
```

Question 2 -

Make a calculator class with a function name “equals” that take an enum case as value like multiply, subtraction, addition, square root, division.

```swift
import UIKit
enum calculate {
    case multiply (Int, Int)
    case subtraction (Int, Int)
    case addition (Int, Int)
    case squareRoot (Float)
    case division (Int, Int)
}
class Calculator {
    func equals (_ equal: calculate)-> Int {
    switch equal {
    case let .multiply(a, b):
        return (a*b)
    case let .subtraction(a,b):
        return (a-b)
    case let .addition(a, b):
        return (a+b)
    case let .division(a, b):
        return (a/b)
    case let .squareRoot(a):
        return (Int(sqrt(a)))
    }
}
}
```

```swift
var obj = Calculator()
let add = obj.equals(.addition(1, 6))
let multi = obj.equals(.multiply(4, 4))
let div = obj.equals(.division(10, 5))
let sqroot = obj.equals(.squareRoot(20))
let subt = obj.equals(.subtraction(33, 3))
print(add)
print(multi)
print(div)
print(sqroot)
print(subt)
```

```
7
16
2
4
30
```

Question 3

Make the same calculator using functions as an argument, define all type functions in a struct.

```swift
import UIKit

enum calculate{
    case double (Double)
    case int (Int)
}

struct calculator{
    var a: Int
    var b: Int
    init(a: Int, b: Int) {
        self.a = a
        self.b = b
    }
    init(a: Int) {
        self.a = 0
        self.b = 0
    }
    func addition(_ add: (Int, Int) -> Int) {
        print("Result is \(add(a,b))")
    }
    func subtraction(_ sub: (Int, Int) -> Int) {
        print("Result is \(sub(a,b))")
    }
    func division(_ div: (Int, Int) -> Int) {
        print("Result is \(div(a,b))")
    }
    func multiplication(_ multiply: (Int, Int) -> Int) {
        print("Result is \(multiply(a,b))")
    }
    func squareRoot(_ root: (Double) -> Double) {
```

```
    func squareRoot(_ root: (Double) -> Double) {
        print("Result is \(root(Double(a)))")
    }
}

    func add(a: Int, b: Int) -> Int {
        return a+b
    }
    func sub(a: Int, b: Int) -> Int {
        return a-b
    }
    func div(a: Int, b: Int) -> Int {
        return a/b
    }
    func multiply(a: Int, b: Int) -> Int {
        return a*b
    }
    func root(a: Double) -> Double {
        return (sqrt(Double(a)))
    }

calculator(a: 10, b: 5).addition(add)
calculator(a: 10, b: 5).subtraction(sub)
calculator(a: 10, b: 5).division(div)
calculator(a: 10, b: 5).multiplication(multiply)
calculator(a: 4).squareRoot(root)
```

```
Result is 15
Result is 5
Result is 2
Result is 50
Result is 0.0
```

Question 4 -

Create a TraineesActivity Class which lazily initializes a data source of all the trainees in an array. Define a closure to filter and find the trainee object based on the name passed. Create an enum explained below which would also have a function returning a closure that takes the trainee object and return a string describing the skill for every enum case. This TraineeActivity would provide three functions as below to perform, record, and rerun the activity. On calling perform passing the name of trainee make use of closure declared to find the trainee object, pass this object to activity closure defined in enum to execute the activity. Later record this activity in any data structure mapped to a trainee and use this data structure to rerun the activity performed.

on deinitialization, it should print - **Hey !!! Thanks, I am gone.**

Note *- Make use of closures, lazy, type alias, optional binding & chaining,*

Outline of class and data should be as follows -

Class TraineesActivity

trainesData - load lazily

closure - chooseFilterObject

functions -

performActivity

recordActivity

rerunActivity

Struct Trainee

- name

- dance = 78

- run = 65

- Sing = 35

- fight = 2

- academic = 46

Enum {

case dance

case academic

case run

case sing

case Fights

a function returning activity closure that take trainee object and prints the activity skill

}

Test Run -

var trainee : Tainees? = Tainees()

trainee?.performActivity(trainee: “Waseem”, activity: .run)

trainee?.performActivity(trainee: “Anindiya”, activity: .academic)

trainee?.performActivity(trainee: “Rekha”, activity: .run)

trainee?.rerunActivity()

trainee = **nil**

Prints log -

Waseem **good run 70**

Anindiya **good academic 45**

No trainee found

Waseem **good run 70**

Anindiya **good academic 45**

Hey !!! Thanks, I am gone.

```swift
import UIKit

enum Activity {
    case dance
    case academic
    case run
    case sing
    case fights

    func enumFuntion() -> String {
        switch self {
        case .dance:
            return " is dancing."
        case .academic:
            return "is studying."
        case .run:
            return "is running."
        case .sing:
            return "is singing."
        case .fights:
            return "is fighting."
        }
    }

    func FilterofEnum(_ traineeName: String, traineeObject: (String) -> Void) {
        traineeObject(traineeName)
    }
}

struct Trainee {
    var name: String
    var dance: Int?
    var run: Int?
    var sing: Int?
    var fight: Int?
    var academic: Int?

}

var trainee: [Trainee] = [Trainee(name: "Waseem", run: 70), Trainee(name: "Anindiya", academic:45 ),Trainee(name: "Rekha", run: 75)]


class TraineeActivity {
    lazy var traineeData: [Trainee] = {
        return trainee }()

    var recordedTrainees: [Trainee] = []

    func performActivity(traineeName name: String, activity en: Activity) {
        var traineeObj: Trainee? = nil
        en.FilterofEnum(name) { (name) in
            for data in traineeData where data.name == name {

                traineeObj = data
            }
        }

        if traineeObj != nil {
            print("\(traineeObj?.name ?? "not") good \(en) \(traineeObj?.run)") ⚠ String interpolation produces a debug description for an optional v
            recordActivity(trainee: traineeObj!)
        }
    }
```

```swift
    func recordActivity(trainee traineeObject: Trainee) {
        recordedTrainees.append(traineeObject)
    }

    func rerunActivity() {
        for item in recordedTrainees {
            print(item)
        }
    }
}

var object = TraineeActivity()

object.performActivity(traineeName: "Waseem", activity: .run)
object.performActivity(traineeName: "Anindiya", activity: .academic)
object.performActivity(traineeName: "Rekha", activity: .run)
object.rerunActivity()

```

```
Waseem good run Optional(70)
Anindiya good academic nil
Rekha good run Optional(75)
Trainee(name: "Waseem", dance: nil, run: Optional(70), sing: nil, fight: nil, academic: nil)
Trainee(name: "Anindiya", dance: nil, run: nil, sing: nil, fight: nil, academic: Optional(45))
Trainee(name: "Rekha", dance: nil, run: Optional(75), sing: nil, fight: nil, academic: nil)
```