Swift Intermediate

Exercise 4

Abhishek Maurya

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]).

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
```

Answer 1

```
var list : [String] = ["green beans", "banana", "black beans", "grapes", "red
beans", "apples", "carrot"]
func siftBeans( fromGroceryList : [String] ) -> (beans : [String] ,
otherGroceries :[String]) {
    var beans = [String]()
    var otherGrocery = [String]()
    for groceryItem in list {
        if (groceryItem.hasSuffix("beans")) {
            beans.append(groceryItem)
        }
        else {
            otherGrocery.append(groceryItem)
        }
    }
    return (beans , otherGrocery)
}
var  result = siftBeans(fromGroceryList: list)
print(result.beans,"are beans.")
print(result.otherGroceries,"are other groceries")
```

Question 2

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

Answer:

```
indirect enum Calculator {
       case number(Float)
       case addition(Calculator, Calculator)
       case substraction(Calculator, Calculator)
```

```
        case multiplication(Calculator, Calculator)
        case division(Calculator, Calculator)
        case sqroot(Calculator)
    }
    func equals(_ operation: Calculator) -> Float {
        switch operation {
        case let .number(value):
            return value
        case let .addition(num1, num2):
            return equals(num1) + equals(num2)
        case let .substraction(num1, num2):
            return equals(num1) - equals(num2)
        case let .multiplication(num1, num2):
            return equals(num1) * equals(num2)
        case let .division(num1, num2):
            return equals(num1) / equals(num2)
        case let .sqroot(num1):
            return sqrt(equals(num1))
        }
    }
    var num1 = (Calculator.number(25))
    var num2 = (Calculator.number(5))
    print("\nFor \(num1) and \(num2)\nAddition is
    \(equals(Calculator.addition(num1, num2)))\nSubtraaction is
    \(equals(Calculator.substraction(num1, num2)))\nMultiplication is
    \(equals(Calculator.multiplication(num1, num2)))\nDivision is
    \(equals(Calculator.division(num1, num2)))\nSquare root is
    \(equals(Calculator.sqroot(num1)))")
```

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

Test Run:

```
let calObj = AnkitCalculator()

calObj.equals(operation: .addition(operandOne: 2, operandTwo: 4))

calObj.equals(operation: .division(operandOne: 5, operandTwo: 4))

// Using Funct as params

calObj.equalFuncOp(operationValues: (4,5), operationFunc:
   AnkitCalculator.CalculatorStruct.addition(operandOne:operandTwo:))

calObj.equalFuncOp(operationValues: (5,7), operationFunc:
   AnkitCalculator.CalculatorStruct.multiplication(operandOne:operandTwo:))
```

Output Result :

Operation Result = 6.0

Operation Result = 3.0

Operation Result using func as param = 9.0

Operation Result using func as param = 25.0

Answer

3. Question 3

Create a TraineesActivity Class which lazily initialise a datasource of all the trainees in an array.

Define a closure to filter and find the trainee object based on name passed .

Create a 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 function 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.

Note - Make use of closures, lazy, typeAlias, optional binding & chaining,

Outline of class and data should be as following -

```
Class TraineesActivity
   trainesData - load lazily
   closure
     choose
   functions -
     performActivity
     recordActivity
     rerunActivity
Struct Trainee
-dace = 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.
```