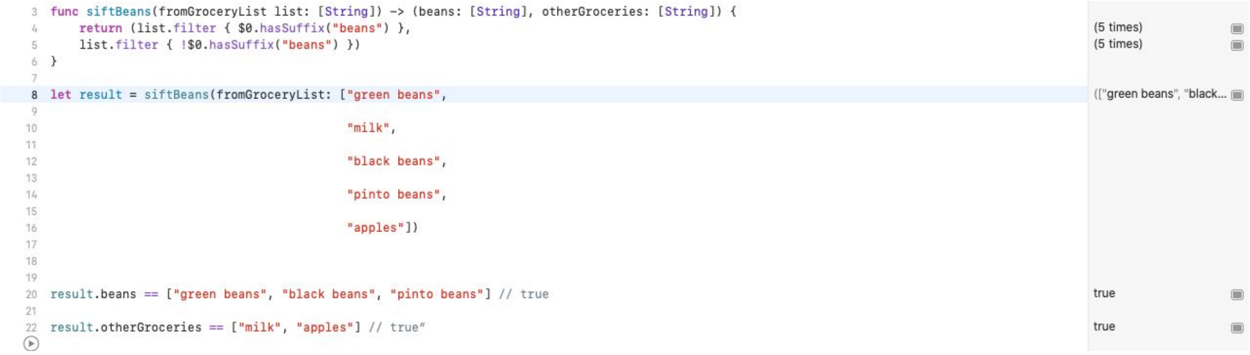
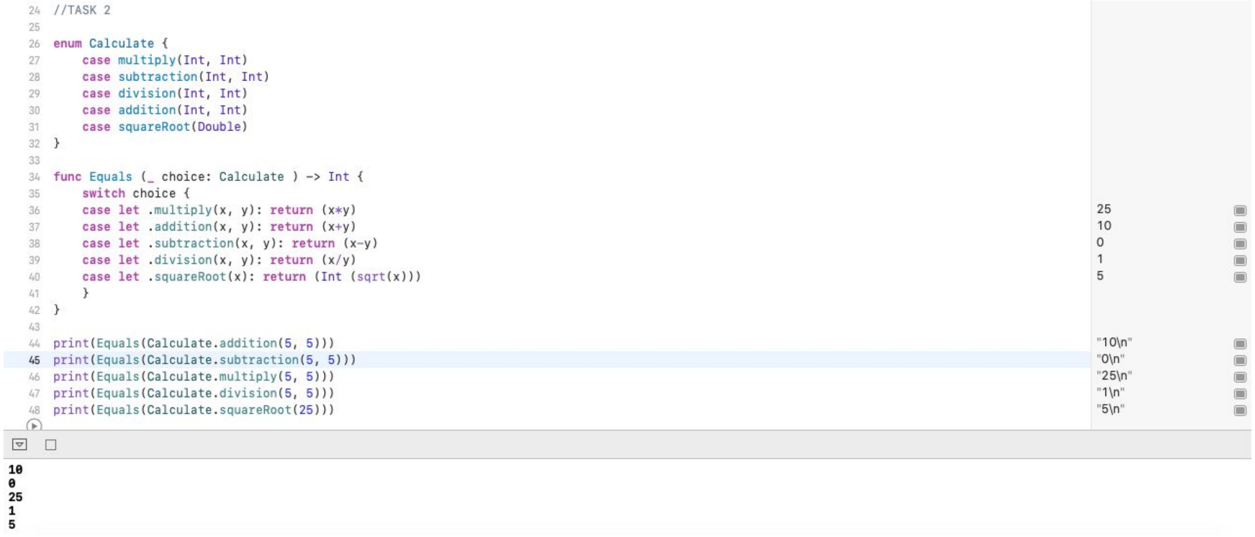
Write a function called siftBeans(fromGroceryList:) that takes a grocery list (as an array of strings) and “sifts out” the beans from the other groceries. The function should take one argument that has a parameter name called list, and it should return a named tuple of the type (beans: [String], otherGroceries: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



```swift
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"] // true

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

```
(5 times)
(5 times)

(["green beans", "black...

true

true
```

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

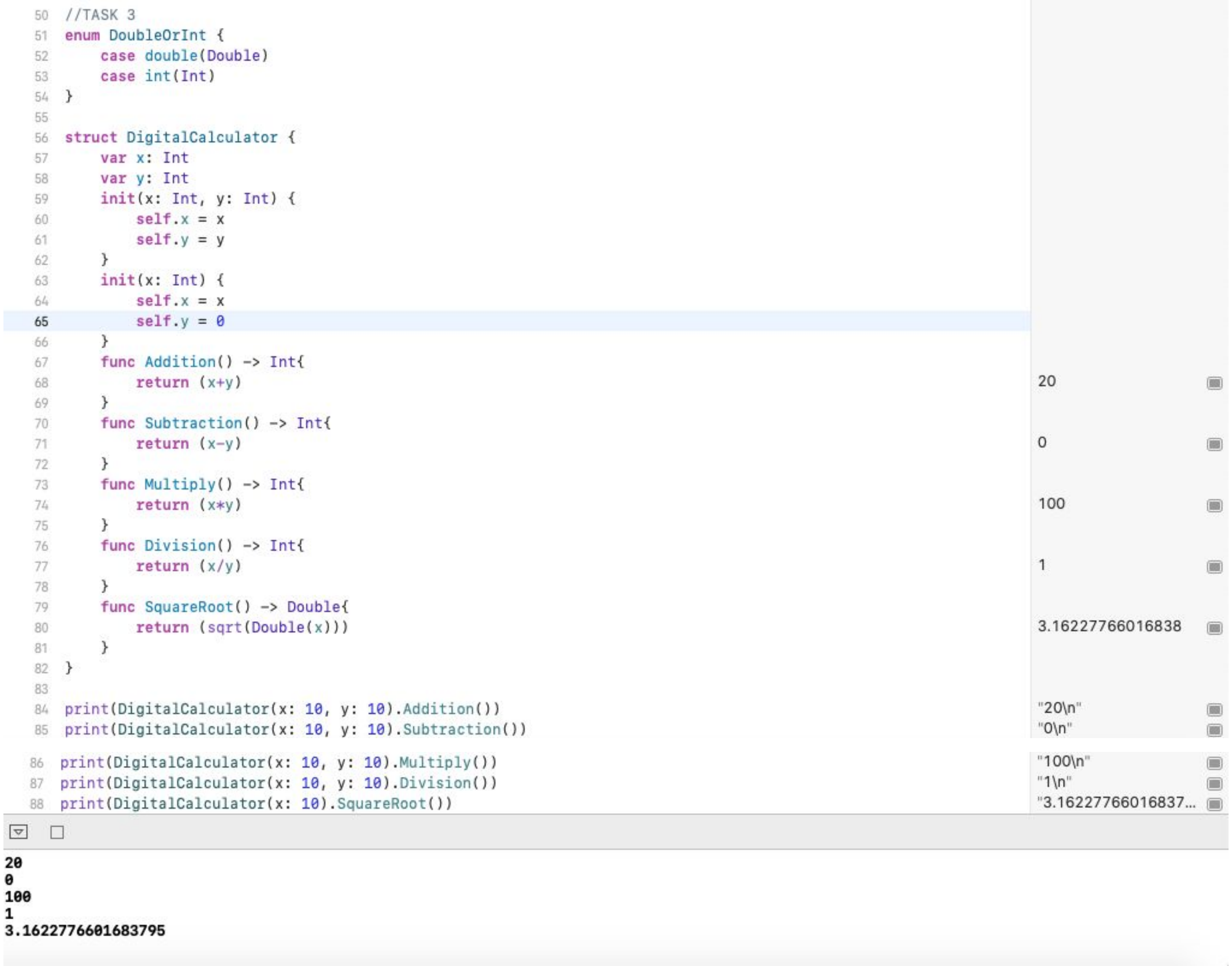


```swift
//TASK 2

enum Calculate {
    case multiply(Int, Int)
    case subtraction(Int, Int)
    case division(Int, Int)
    case addition(Int, Int)
    case squareRoot(Double)
}

func Equals (_ choice: Calculate ) -> Int {
    switch choice {
    case let .multiply(x, y): return (x*y)
    case let .addition(x, y): return (x+y)
    case let .subtraction(x, y): return (x-y)
    case let .division(x, y): return (x/y)
    case let .squareRoot(x): return (Int (sqrt(x)))
    }
}

print(Equals(Calculate.addition(5, 5)))
print(Equals(Calculate.subtraction(5, 5)))
print(Equals(Calculate.multiply(5, 5)))
print(Equals(Calculate.division(5, 5)))
print(Equals(Calculate.squareRoot(25)))
```

```
25
10
0
1
5

"10\n"
"0\n"
"25\n"
"1\n"
"5\n"
```

```
10
0
25
1
5
```

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



```swift
//TASK 3
enum DoubleOrInt {
    case double(Double)
    case int(Int)
}

struct DigitalCalculator {
    var x: Int
    var y: Int
    init(x: Int, y: Int) {
        self.x = x
        self.y = y
    }
    init(x: Int) {
        self.x = x
        self.y = 0
    }
    func Addition() -> Int{
        return (x+y)
    }
    func Subtraction() -> Int{
        return (x-y)
    }
    func Multiply() -> Int{
        return (x*y)
    }
    func Division() -> Int{
        return (x/y)
    }
    func SquareRoot() -> Double{
        return (sqrt(Double(x)))
    }
}

print(DigitalCalculator(x: 10, y: 10).Addition())
print(DigitalCalculator(x: 10, y: 10).Subtraction())
print(DigitalCalculator(x: 10, y: 10).Multiply())
print(DigitalCalculator(x: 10, y: 10).Division())
print(DigitalCalculator(x: 10).SquareRoot())
```

```
20
0
100
1
3.16227766016838
"20\n"
"0\n"
"100\n"
"1\n"
"3.16227766016837...
```

```
20
0
100
1
3.1622776601683795
```

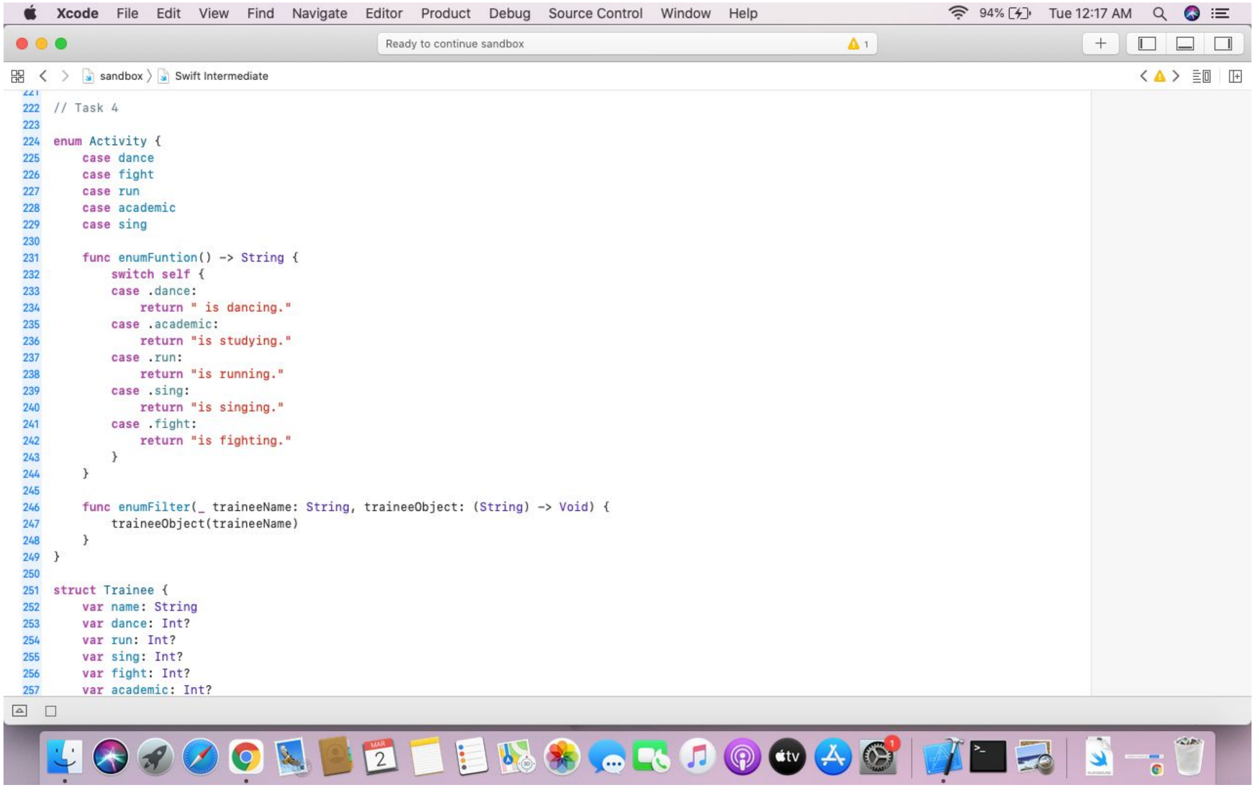
Create a TraineesActivity Class which lazily initializes a data source of all the trainees in an array.

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

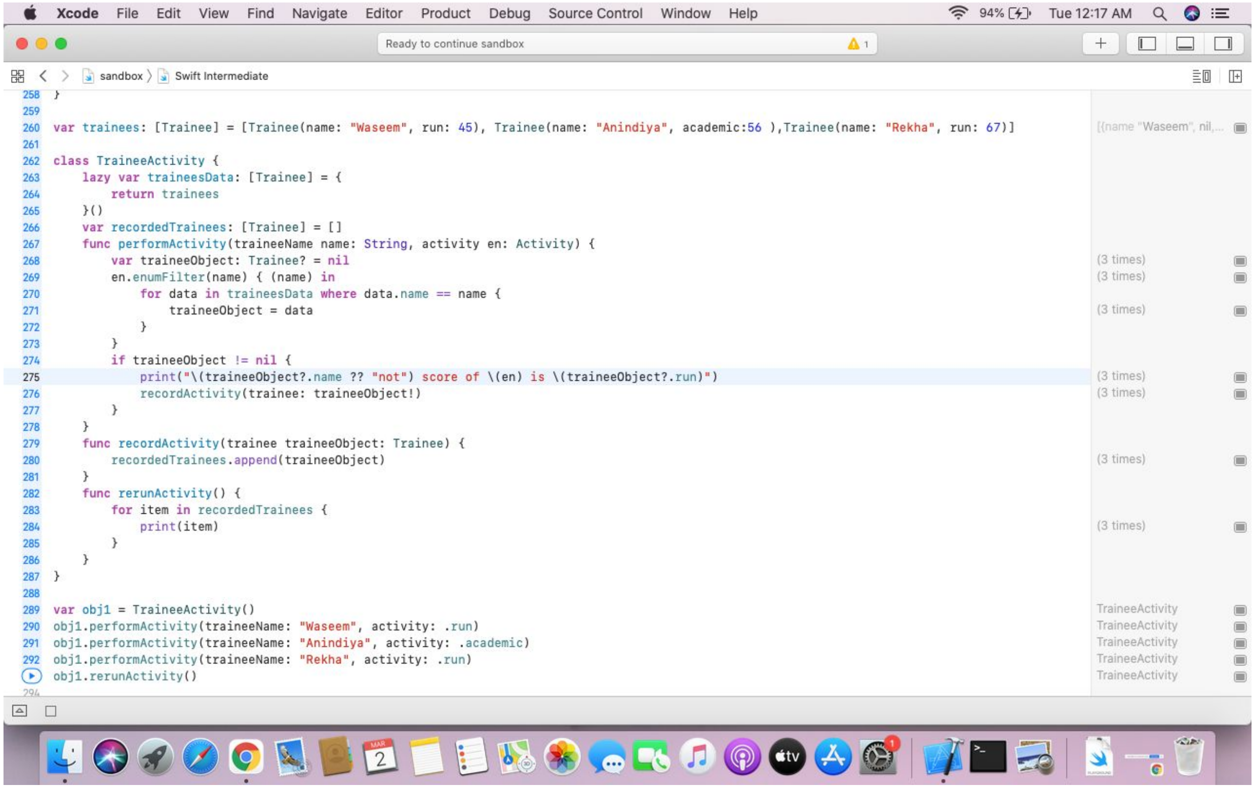
Create an enum explained below which would also have a function returning a closure that takes the trainee object and return a string describing the skill for every enum case.

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


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
94%
Tue 12:17 AM
Ready to continue sandbox
1
sandbox
Swift Intermediate

```swift
// Task 4

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

    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 .fight:
            return "is fighting."
        }
    }

    func enumFilter(_ 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?
```



```swift
}

var trainees: [Trainee] = [Trainee(name: "Waseem", run: 45), Trainee(name: "Anindiya", academic:56 ),Trainee(name: "Rekha", run: 67)]

class TraineeActivity {
    lazy var traineesData: [Trainee] = {
        return trainees
    }()
    var recordedTrainees: [Trainee] = []
    func performActivity(traineeName name: String, activity en: Activity) {
        var traineeObject: Trainee? = nil
        en.enumFilter(name) { (name) in
            for data in traineesData where data.name == name {
                traineeObject = data
            }
        }
        if traineeObject != nil {
            print("\(traineeObject?.name ?? "not") score of \(en) is \(traineeObject?.run)")
            recordActivity(trainee: traineeObject!)
        }
    }
    func recordActivity(trainee traineeObject: Trainee) {
        recordedTrainees.append(traineeObject)
    }
    func rerunActivity() {
        for item in recordedTrainees {
            print(item)
        }
    }
}

var obj1 = TraineeActivity()
obj1.performActivity(traineeName: "Waseem", activity: .run)
obj1.performActivity(traineeName: "Anindiya", activity: .academic)
obj1.performActivity(traineeName: "Rekha", activity: .run)
obj1.rerunActivity()
```

```
Waseem score of run is Optional(45)
Anindiya score of academic is nil
Rekha score of run is Optional(67)
Trainee(name: "Waseem", dance: nil, run: Optional(45), sing: nil, fight: nil, academic: nil)
Trainee(name: "Anindiya", dance: nil, run: nil, sing: nil, fight: nil, academic: Optional(56))
Trainee(name: "Rekha", dance: nil, run: Optional(67), sing: nil, fight: nil, academic: nil)
```