iOS : Swift Intermediate level

Name : Shreyansh Raj Keshri
Date : 19/02/2020

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]).

Ans.
```
func siftBeans(fromGroceryList list: [String]) -> (beans: [String], otherGroceries: [String]) {
    var beans = [String]()
    var otherGroceries = [String]()

    for product in list {
        if product.hasSuffix("beans") {
            beans.append(product)
        } else {
            otherGroceries.append(product)
        }
    }

    return (beans, otherGroceries)
}

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

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

```
class Calculator{
```

```
    enum Operations {
        case addition(Int, Int)
        case subtraction(Int, Int)
        case multiplication(Int, Int)
        case division(Int, Int)
    }

    func equals(enumArgument : Operations) -> Int {
        switch enumArgument {
        case .addition(let a, let b):
            return a + b
        case .subtraction(let a, let b):
            return a - b
        case .multiplication(let a, let b):
            return a*b
        case .division(let a, let b):
            return a/b
        }
    }

    func equals(argumentAs function : (Int, Int) -> Int, num1 : Int, num2 : Int) -> Int
{
        return function(num1, num2)
    }


}
var abc = Calculator()
let testCaseForQuestion2 = abc.equals(enumArgument: .addition(1, 2))
```

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

```
struct Operation {

    static func addition(a : Int, b : Int) -> Int {
```

```
        return a + b
    }

    static func subtraction(a : Int, b : Int) -> Int {
        return a - b
    }

    static func multiplication(a : Int, b : Int) -> Int {
        return a * b
    }

    static func division(a : Int, b : Int) -> Int {
        return a / b
    }

}


let abc2 = abc.equals(argumentAs: Operation.multiplication(a:b:), num1: 1, num2: 2)
```