

Date: 24.02.2021

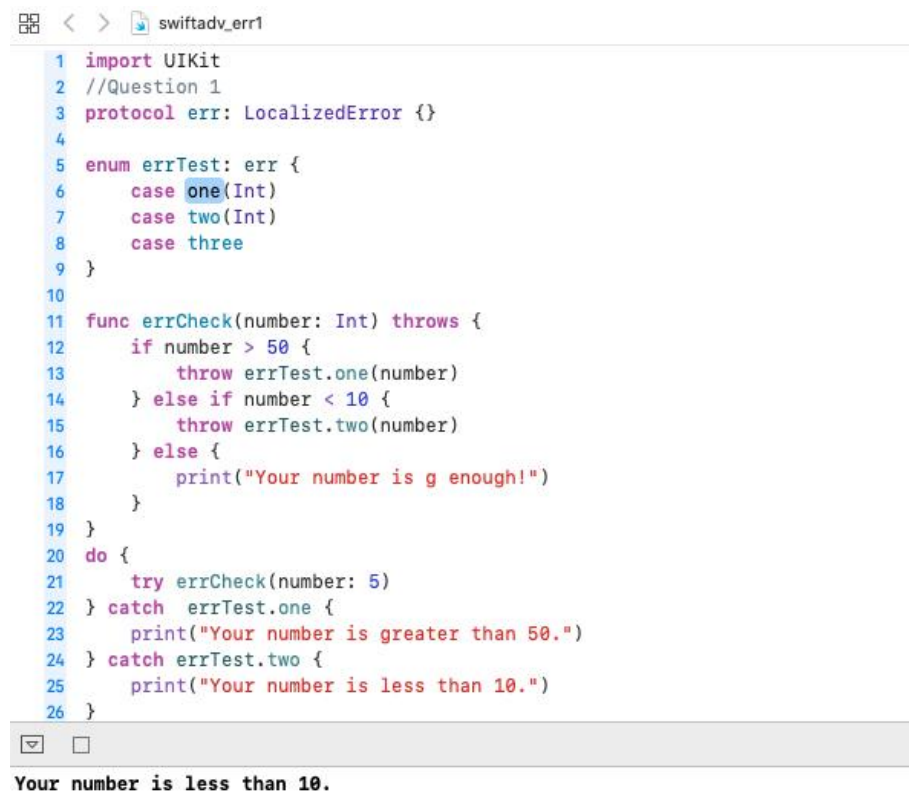
Kavya Casshyap
Mentor: Mr. Amit Kumar
SPOC: Mr. Abhishek Maurya

SWIFT ADVANCE / ERROR HANDLING Assignment

Question 1:

What is Error Protocol. Create a custom error conforming to error protocol.

Error Protocol is just a type for representing error values that can be thrown. Swift creates custom Error type. Typically an Enum is used which conforms to the Error Protocol.



```
1 import UIKit
2 //Question 1
3 protocol err: LocalizedError {}
4
5 enum errTest: err {
6     case one(Int)
7     case two(Int)
8     case three
9 }
10
11 func errCheck(number: Int) throws {
12     if number > 50 {
13         throw errTest.one(number)
14     } else if number < 10 {
15         throw errTest.two(number)
16     } else {
17         print("Your number is g enough!")
18     }
19 }
20 do {
21     try errCheck(number: 5)
22 } catch errTest.one {
23     print("Your number is greater than 50.")
24 } catch errTest.two {
25     print("Your number is less than 10.")
26 }
```

Your number is less than 10.

Question 2:

Write a failable initialiser for a class which throws a error "Object not able to initialise" description a initialisationFailed case, Catch the error and print its error description.

```

28 //Question 2
29
30 enum FriendsChar: String {
31     case one = "Joey"
32     case two = "Ross"
33 }
34 enum errName: Swift.Error {
35     case initializationFailed
36 }
37
38 struct Habits {
39     var friendschar: FriendsChar = .one
40     init?(word: String) throws {
41         guard let centralperk = FriendsChar(rawValue: word) else {
42             throw errName.initializationFailed
43         }
44         self.friendschar = centralperk
45     }
46 }
47
48 do {
49     _ = try Habits(word: "")
50 } catch errName.initializationFailed {
51     print("Object not able to initialise.")
52 }

```



Object not able to initialise.

Question 3:

Explain the difference try, try?, try! , make sure to write a program to explain the difference.

In **Try?** Keyword an error is thrown, the error is handled by turning it into an optional value. This means that there is no need to wrap the throwing method call in a do-catch statement.

If the operation fails, the method returns an optional without a value. If the operation succeeds, the optional contains a value.

```

55 //Question 3
56
57 enum FriendsChar: String {
58     case joey = "joey"
59     case ross = "ross"
60 }
61 enum errName: Swift.Error {
62     case initializationFailed
63 }
64
65 struct profession {
66     var friendschar: FriendsChar = .joey
67     init?(word: String) throws {
68         guard let centralperk = FriendsChar(rawValue: word) else {
69             throw errName.initializationFailed
70         }
71         self.friendschar = centralperk
72     }
73 }
74 let doctor = try? profession(word: "")
75 let actor = try? profession(word: "joey")
76 let chef = try! profession(word: "")

```



Fatal error: 'try!' expression unexpectedly raised an error: __lldb_expr_9.errName.initializationFailed: file swiftadv_err1.playground, line 76

Try! – It is dangerous to use try! By appending an exclamation mark to the try keyword, error propagation is disabled. This means that, if an error does get thrown, the application crashes as the result of a runtime error.

```
33 enum ThrowableError: Error {
34     case badError(howBad: Int)
35 }
36
37 func doSomething(everythingIsFine: Bool = false) throws -> String {
38     if everythingIsFine {
39         return "Everything is ok"
40     } else {
41         throw ThrowableError.badError(howBad: 4)
42     }
43 }
44
45
46
47 func doSomeOtherThing() throws -> Void {
48     // Not within a do-catch block.
49     // Any errors will be re-thrown to callers.
50     _ = try doSomething()
51 }
52
53
54 let result = try! doSomething()
55
56 if (try? doSomething()) != nil {
57     // doSomething succeeded, and result is unwrapped.
58 } else {
59     // Ouch, doSomething() threw an error.
60 }
61 }
```

error: Execution was interrupted, reason: EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0).

Fatal error: 'try!' expression unexpectedly raised an error: __lldb_expr_22.ThrowableError.badError(howBad: 4): file swifterr3.playground, line 54

Try - The try/catch syntax was added in Swift 2.0 to make exception handling clearer and safer. It's made up of three parts: do starts a block of code that might fail, catch is where execution gets transferred if any errors occur, and any function calls that might fail need to be called using try.

```
1 import UIKit
2
3 enum MagicWords: String {
4     case abracadbra = "abracadabra"
5     case alakazam = "alakazam"
6 }
7
8 enum NameError: Swift.Error {
9     case noName
10 }
11
12 struct Spellz {
13     var magicWords: MagicWords = .abracadbra
14
15     init?(words: String) throws {
16         guard let incantation = MagicWords(rawValue: words) else {
17             throw NameError.noName
18         }
19         self.magicWords = incantation
20     }
21 }
22 do {
23     _ = try Spellz(words: "")
24 } catch NameError.noName {
25     print("No name found!!")
26 }
27
28 let dangerousMagicSpell = try? Spellz(words: "") //return nil
29 let safeMagicSpell = try? Spellz(words: "alakazam") //works
30
```

No name found!!

Question 4:

Write a program which loads the data from a datasource of 10 employees looks like below, Program would help to give salary bonus to employees. Which is based on some conditions but if employee is not able to satisfy the condition program should throw the error with specific error condition and its description should be printed.

Conditions -

Employee should be present on the day.

Employee has completed a year with company

Employee competency should be hot competency like ios, android, bigdata, AI etc. (make some placeholder competency which would not get bonus)

Employee should have a attendance percent over 80.

Emp -

empID

empName

empEmail

joiningDate

isPresent

competency

attendancePercent

Test Run:

```
let bonusProgram = BonusProgram()
```

```
bonusProgram.allowedForBonus("muskaan @tothenew.com")
```

```
bonusProgram.allowedForBonus("mithlesh @tothenew.com")
```

```
bonusProgram.allowedForBonus("ankit @tothenew.com")
```

```
bonusProgram.allowedForBonus("sachin @tothenew.com")
```

bonusProgram.allowedForBonus("Merry@tothenew.com")

*.... few more run-
// prints*

Muskaan is absent today.

Mithlesh competency does not fall under bonus program.

ankit is eligible for bonus.

sachin has joined us on dd/mm/yyyy and still to complete a year with us.

Merry is eligible for bonus.

//Question 4

```
struct employee{
    var empID : Int
    var empName : String
    var empEmail : String
    var yearsOfExperience : Double
    var isPresent : Bool
    var competency : String
    var attendancePercentage : Int
}

let empInfo: [employee] = [employee(empID: 101, empName: "Joey", empEmail: "joey@tothenew.com", yearsOfExperience: 3, isPresent: true,
    competency: "iOS", attendancePercentage: 90),

    employee(empID: 102, empName: "Ross", empEmail: "ross@tothenew.com", yearsOfExperience: 2, isPresent: true,
        competency: "iOS", attendancePercentage: 85),

    employee(empID: 103, empName: "Rachel", empEmail: "rachel@tothenew.com", yearsOfExperience: 4.5, isPresent:
        true, competency: "AI", attendancePercentage: 89),

    employee(empID: 104, empName: "Phoebe", empEmail: "phoebe@tothenew.com", yearsOfExperience: 1, isPresent:
        true, competency: "Android", attendancePercentage: 75),

    employee(empID: 105, empName: "Chandler", empEmail: "chandler@tothenew.com", yearsOfExperience: 5, isPresent:
        false, competency: "android", attendancePercentage: 97),

    employee(empID: 106, empName: "Monica", empEmail: "monica@tothenew.com", yearsOfExperience: 6, isPresent:
        true, competency: "BigData", attendancePercentage: 70),

    employee(empID: 107, empName: "Mithlesh", empEmail: "mithlesh@tothenew.com", yearsOfExperience: 1.5,
        isPresent: true, competency: "BigData", attendancePercentage: 81),

    employee(empID: 108, empName: "Merry", empEmail: "merry@tothenew.com", yearsOfExperience: 1.8, isPresent:
        true, competency: "iOS", attendancePercentage: 85),

    employee(empID: 109, empName: "Ankit", empEmail: "ankit@tothenew.com", yearsOfExperience: 2, isPresent: true,
        competency: "QE", attendancePercentage: 75),

    employee(empID: 110, empName: "Sachin", empEmail: "sachin@tothenew.com", yearsOfExperience: 3.5, isPresent:
        true, competency: "AI", attendancePercentage: 87)]

enum bonusErr: Error{

    case isPresentorNot (String)
    case yeo (String)
    case CompetencyNotHot (String)
    case attendance (String)
    var localDescription: String{

        switch self {

            case .isPresentorNot(let name):

                return name

            case .yeo(let name):

                return name

            case .CompetencyNotHot(let name):

                return name

            case .attendance(let name):

                return name

        }
    }
}

class bonus {

    func BonusAllowed(bonusemail : String) throws -> Void {
        var item: employee {
            var obj: employee? = nil
            for ob in empInfo where bonusemail == ob.empEmail {
                obj = ob
            }
            return obj!
        }
    }
}
```

```

        if item.isPresent == true{
            if item.yearsOfExperience > 1 {
                if item.competency == "iOS" || item.competency == "android" || item.competency == "BigData" || item.competency == "AI"{
                    if item.attendancePercentage > 80 {
                        print("\(item.empName) is eligible for bonus!")
                    } else {
                        throw bonusErr.attendance(item.empName)
                    }
                } else {
                    throw bonusErr.CompetencyNotHot(item.empName)
                }
            }
        } else {
            throw bonusErr.yeo(item.empName)
        }
    }
    else {
        throw bonusErr.isPresentorNot(item.empName)
    }
}

let bonusProgram = bonus()

do {

    let _ = try bonusProgram.BonusAllowed(bonusemail: "mithlesh@tothenew.com")

    let _ = try bonusProgram.BonusAllowed(bonusemail: "merry@tothenew.com")

    let _ = try bonusProgram.BonusAllowed(bonusemail: "rachel@tothenew.com")

    let _ = try bonusProgram.BonusAllowed(bonusemail: "phoebe@tothenew.com")

    let _ = try bonusProgram.BonusAllowed(bonusemail: "ankit@tothenew.com")

189 } catch bonusErr.isPresentorNot(let name){
190
191     print(name + " is absent today")
192
193 } catch bonusErr.CompetencyNotHot(let name){
194
195     print(name + " competency does not fall under bonus program.")
196
197 } catch bonusErr.yeo(let name){
198
199     print(name + " is still to complete a year with us")
200
201 } catch bonusErr.attendance(let name) {
202
203     print(name + " has attendance less than 80")
204

```



```

Mithlesh is eligible for bonus!
Merry is eligible for bonus!
Rachel is eligible for bonus!
Phoebe is still to complete a year with us

```