

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. Typically an Enum is used which conforms to the Error Protocol. The Error Protocol is more or less empty.

```
1 import UIKit
2 //Task 1
3 protocol Error: LocalizedError {}
4
5 enum ErrorTest: Error{
6     case moreAmount(Int)
7     case lessAmount(Int)
8     case okAmount
9 }
10
11 func checkAmountWithError(amount: Int) throws {
12     if amount > 2000 {
13         throw ErrorTest.moreAmount(amount)
14     } else if amount < 1100 {
15         throw ErrorTest.lessAmount(amount)
16     } else {
17         print("You have enough amount")
18     }
19 }
20
21 do {
22     try checkAmountWithError(amount: 350)
23 } catch ErrorTest.moreAmount {
24     print("You have more money than needed")
25 } catch ErrorTest.lessAmount {
26     print("Bring more money")
27 }
```

Bring more money\n

Bring more money

Question 2: Write a failable initializer for a class which throws an error “Object not able to initialise” description a initialisationFailed case, Catch the error and print its error description.

```
28 //Task 2
29 enum MagicWords: String {
30     case abracadabra = "abracadabra"
31     case alakazam = "alakazam"
32 }
33 enum NameError: Swift.Error {
34     case initialisationFailed
35 }
36
37 struct Spellz {
38     var magicWords: MagicWords = .abracadabra
39     init?(words: String) throws {
40         guard let incantation = MagicWords(rawValue: words) else {
41             throw NameError.initialisationFailed
42         }
43         self.magicWords = incantation
44     }
45 }
46
47 do {
48     _ = try Spellz(words: "")
49 } catch NameError.initialisationFailed {
50     print("Object not able to initialise")
51 }
```

Object not able to initi...

Object not able to initialise

Question 3: Explain the difference try, try?, try! , make sure to write a program to explain the difference.

try keyword is used to handle the errors with the help of do-catch block. Where try is used to execute a statement which is suspected to throw and error in the do block.

try? keyword and 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.

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

```
28 //Task 2
29 enum MagicWords: String {
30     case abracadbra = "abracadabra"
31     case alakazam = "alakazam"
32 }
33 enum NameError: Swift.Error {
34     case initialisationFailed
35 }
36
37 struct Spellz {
38     var magicWords: MagicWords = .abracadbra
39     init?(words: String) throws {
40         guard let incantation = MagicWords(rawValue: words) else {
41             throw NameError.initialisationFailed
42         }
43         self.magicWords = incantation
44     }
45 }
46
47 // Task 3
48 let dangerousMagicSpell = try? Spellz(words: "") //return nil
49 let safeMagicSpell = try? Spellz(words: "alakazam") //works
50 let fatalMagicSpell = try! Spellz(words: "")
51
52
53
```

nil
Spellz

Fatal error: 'try!' expression unexpectedly raised an error: _lldb_expr_26.NameError.initialisationFailed: file Swift Advance-Error Handling.xcplaygroundpage, line 56
Playground execution failed:

error: Execution was interrupted, reason: EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0).
The process has been left at the point where it was interrupted, use "thread return -x" to return to the state before expression evaluation.

* thread #1, queue = 'com.apple.main-thread', stop reason = EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0)
* frame #0: 0x0000000109c74099 libswiftCore.dylib`Swift.assertionFailure(: Swift.StaticString, : Swift.String, file: Swift.StaticString, line: Swift.UInt, flags:

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 the employee is not able to satisfy the condition program should throw the error with specific error condition and its description should be printed.

```

60 //Task 4
61 struct employee{
62     var empID : Int
63     var empName : String
64     var empEmail : String
65     var yearOfExperience : Double
66     var isPresent : Bool
67     var competency : String
68     var attendancePercent : Int
69 }
70
71
72 let employeeData: [employee] = [employee(empID: 1, empName: "Aryan", empEmail: "aryan@tothenew.com", yearOfExperience: 2.5, isPresent: true,
73 competency: "iOS", attendancePercent: 95),
74 employee(empID: 3, empName: "Kavya", empEmail: "kavya@tothenew.com", yearOfExperience: 1.5, isPresent: true,
75 competency: "iOS", attendancePercent: 98),
76 employee(empID: 4, empName: "Rahul", empEmail: "rahul@tothenew.com", yearOfExperience: 2.5, isPresent: true,
77 competency: "AI", attendancePercent: 92),
78 employee(empID: 2, empName: "Rhythm", empEmail: "rhythm@tothenew.com", yearOfExperience: 0.8, isPresent: true,
79 competency: "android", attendancePercent: 88),
80 employee(empID: 5, empName: "Harsh", empEmail: "harsh@tothenew.com", yearOfExperience: 4.0, isPresent: false,
81 competency: "android", attendancePercent: 90),
82 employee(empID: 6, empName: "Vijender", empEmail: "vijender@tothenew.com", yearOfExperience: 1.5, isPresent:
83 true, competency: "BigData", attendancePercent: 76),
84 employee(empID: 7, empName: "Mithilesh", empEmail: "mithilesh@tothenew.com", yearOfExperience: 1.1, isPresent:
85 true, competency: "BigData", attendancePercent: 81),
86 employee(empID: 8, empName: "Merry", empEmail: "merry@tothenew.com", yearOfExperience: 1.5, isPresent: true,
87 competency: "iOS", attendancePercent: 90),
88 employee(empID: 9, empName: "Aniket", empEmail: "aniket@tothenew.com", yearOfExperience: 2.0, isPresent: true,
89 competency: "QE", attendancePercent: 90),
90 employee(empID: 10, empName: "Sachin", empEmail: "sachin@tothenew.com", yearOfExperience: 3.5, isPresent:
91 true, competency: "AI", attendancePercent: 85)]

```

{empID 1, empName "...

```

85 enum bonusError: Error{
86     case isPresent (String)
87     case yeo (String)
88     case nothotCompetency (String)
89     case attendance (String)
90     var localizedDescription: String{
91         switch self {
92             case .isPresent(let name):
93                 return name
94             case .yeo(let name):
95                 return name
96             case .nothotCompetency(let name):
97                 return name
98             case .attendance(let name):
99                 return name
100         }
101     }
102 }
103
104
105 // inside class, parameter email,
106 class bonus {
107     func allowedForBonus(bonusemail : String) throws -> Void {
108         var item: employee {
109             var object: employee? = nil
110             for ob in employeeData where bonusemail == ob.empEmail {
111                 object = ob
112             }
113             return object!
114         }
115         if item.isPresent == true{
116             if item.yearOfExperience > 1.0{
117                 if item.competency == "iOS" || item.competency == "android" || item.competency == "BigData" || item.competency == "AI"{
118                     if item.attendancePercent > 80{
119                         print("\(item.empName) is eligible for bonus")
120                     }else{
121                         throw bonusError.attendance(item.empName)
122                     }
123                 }else{
124                     throw bonusError.nothotCompetency(item.empName)
125                 }
126             }
127             else{
128                 throw bonusError.yeo(item.empName)
129             }
130         }
131         else{
132             throw bonusError.isPresent(item.empName)
133         }
134     }
135 }
136
137 //let result = try allowedForBonus(bonusdata: employeeData)
138
139 let bonusProgram = bonus()
140
141 do {
142     let _ = try bonusProgram.allowedForBonus(bonusemail: "mithilesh@tothenew.com")
143     let _ = try bonusProgram.allowedForBonus(bonusemail: "aryan@tothenew.com")
144     let _ = try bonusProgram.allowedForBonus(bonusemail: "merry@tothenew.com")
145     let _ = try bonusProgram.allowedForBonus(bonusemail: "rahul@tothenew.com")
146     let _ = try bonusProgram.allowedForBonus(bonusemail: "harsh@tothenew.com")
147     let _ = try bonusProgram.allowedForBonus(bonusemail: "aniket@tothenew.com")
148 } catch bonusError.isPresent(let name){
149     print(name + " is absent today")
150 } catch bonusError.nothotCompetency(let name){
151     print(name + " competency does not fall under bonus program")
152 } catch bonusError.yeo(let name){
153     print(name + " is still to complete a year with us")
154 } catch bonusError.attendance(let name) {
155     print(name + " has attendance less than 80")
156 }

```

Mithilesh is eligible for bonus
Aryan is eligible for bonus
Merry is eligible for bonus
Rahul is eligible for bonus
Harsh is absent today