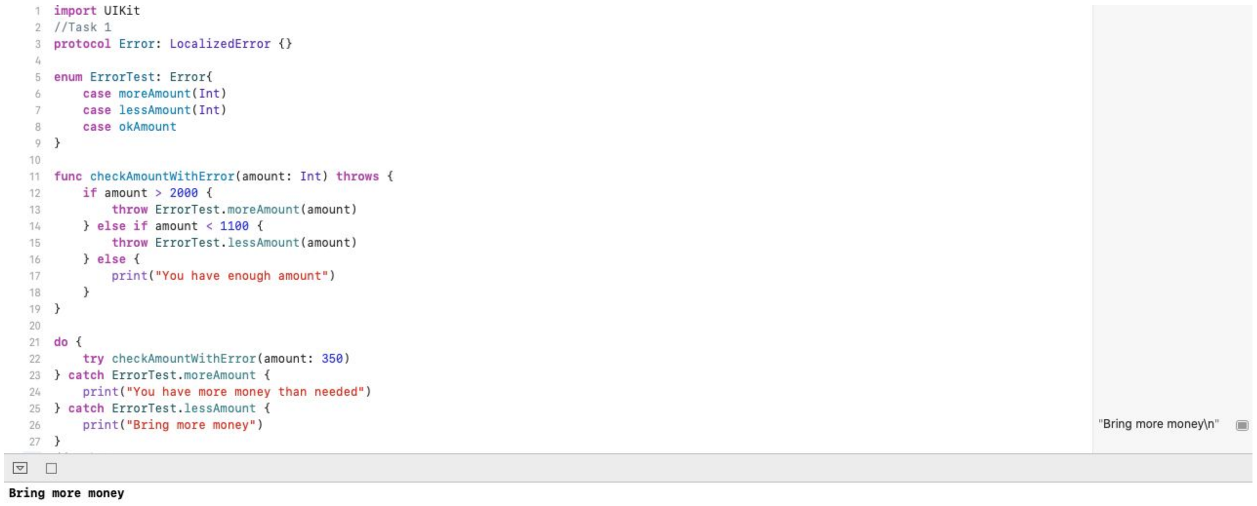
Question 1: What is Error Protocol. Create a custom error conforming to error protocol.

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.

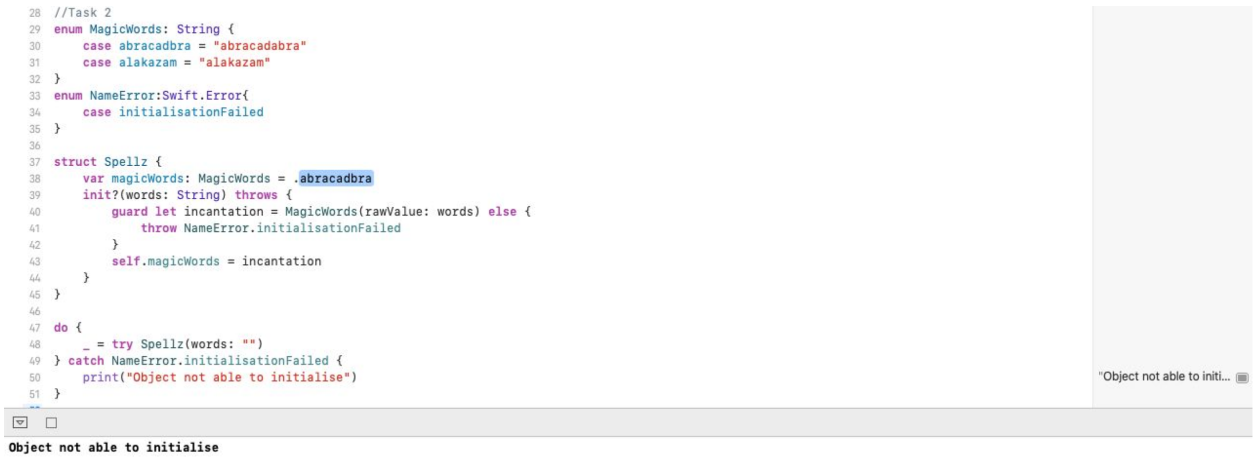


```swift
import UIKit
//Task 1
protocol Error: LocalizedError {}

enum ErrorTest: Error{
    case moreAmount(Int)
    case lessAmount(Int)
    case okAmount
}

func checkAmountWithError(amount: Int) throws {
    if amount > 2000 {
        throw ErrorTest.moreAmount(amount)
    } else if amount < 1100 {
        throw ErrorTest.lessAmount(amount)
    } else {
        print("You have enough amount")
    }
}

do {
    try checkAmountWithError(amount: 350)
} catch ErrorTest.moreAmount {
    print("You have more money than needed")
} catch ErrorTest.lessAmount {
    print("Bring more money")
}
```

"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.



```swift
//Task 2
enum MagicWords: String {
    case abracadbra = "abracadabra"
    case alakazam = "alakazam"
}
enum NameError:Swift.Error{
    case initialisationFailed
}

struct Spellz {
    var magicWords: MagicWords = .abracadbra
    init?(words: String) throws {
        guard let incantation = MagicWords(rawValue: words) else {
            throw NameError.initialisationFailed
        }
        self.magicWords = incantation
    }
}

do {
    _ = try Spellz(words: "")
} catch NameError.initialisationFailed {
    print("Object not able to initialise")
}
```

"Object not able to initi...

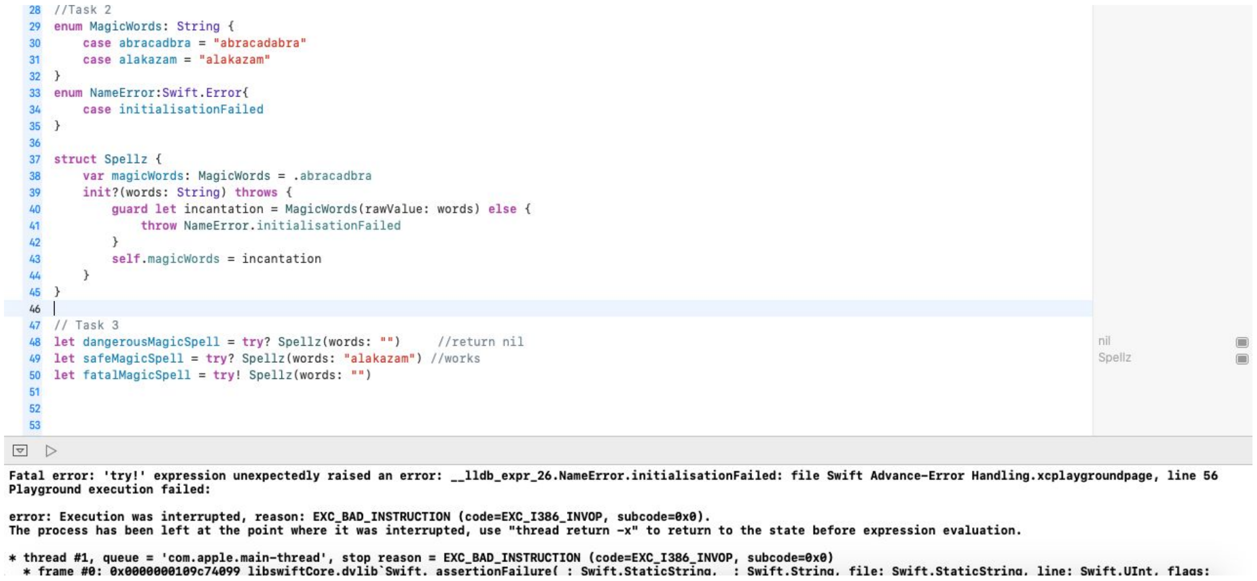
```
Object not able to initialise
```

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.

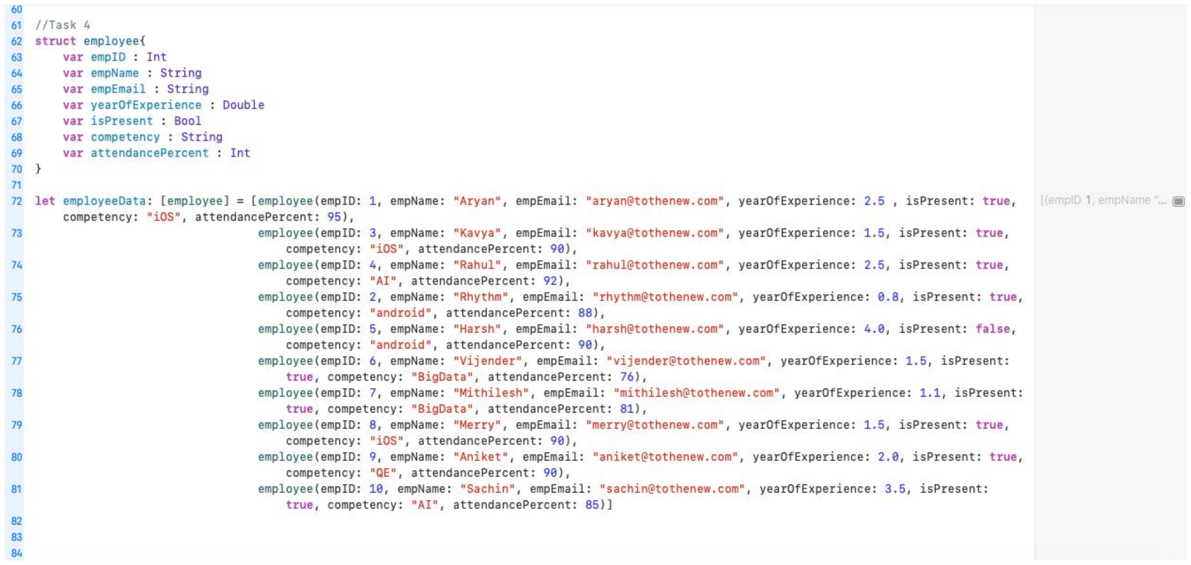


```swift
//Task 2
enum MagicWords: String {
    case abracadbra = "abracadabra"
    case alakazam = "alakazam"
}
enum NameError:Swift.Error{
    case initialisationFailed
}

struct Spellz {
    var magicWords: MagicWords = .abracadbra
    init?(words: String) throws {
        guard let incantation = MagicWords(rawValue: words) else {
            throw NameError.initialisationFailed
        }
        self.magicWords = incantation
    }
}

// Task 3
let dangerousMagicSpell = try? Spellz(words: "")     //return nil
let safeMagicSpell = try? Spellz(words: "alakazam") //works
let fatalMagicSpell = try! Spellz(words: "")
```

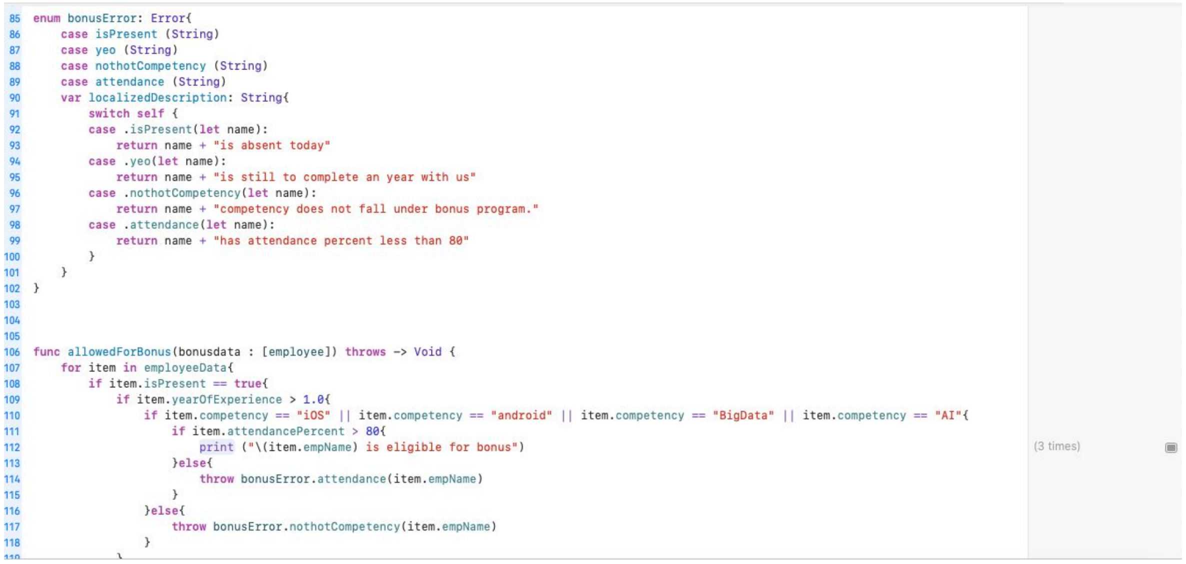
```
nil
Spellz
```

```
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:
```

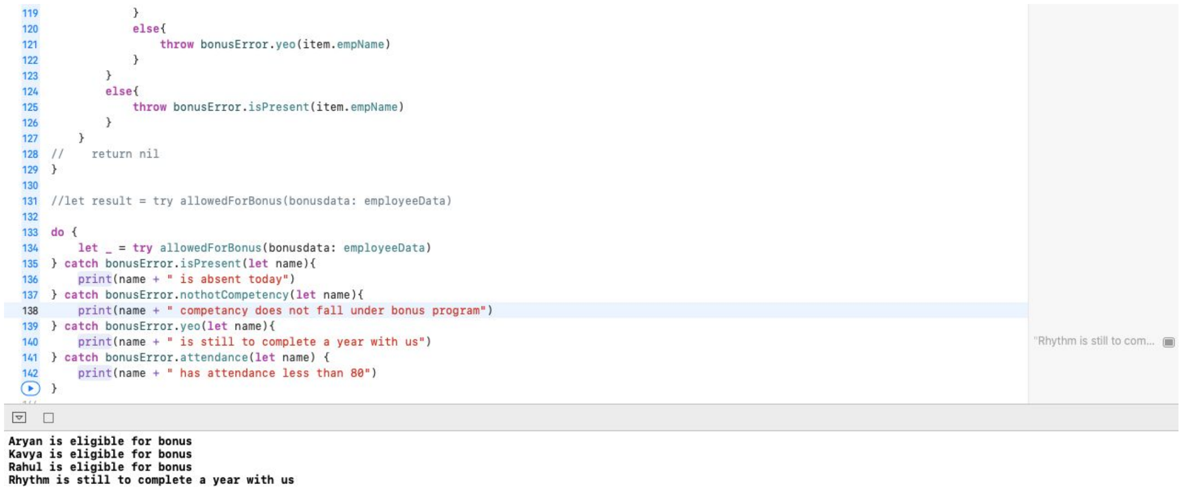
Question 4: Write a program which loads the data from a datasource of 10 employees looks like below, Program would help to give salary bonus to employees. Which is based on some conditions but if the employee is not able to satisfy the condition program should throw the error with specific error condition and its description should be printed.



```swift

//Task 4
struct employee{
    var empID : Int
    var empName : String
    var empEmail : String
    var yearOfExperience : Double
    var isPresent : Bool
    var competency : String
    var attendancePercent : Int
}

let employeeData: [employee] = [employee(empID: 1, empName: "Aryan", empEmail: "aryan@tothenew.com", yearOfExperience: 2.5 , isPresent: true,
    competency: "iOS", attendancePercent: 95),
                                employee(empID: 3, empName: "Kavya", empEmail: "kavya@tothenew.com", yearOfExperience: 1.5, isPresent: true,
                                    competency: "iOS", attendancePercent: 90),
                                employee(empID: 4, empName: "Rahul", empEmail: "rahul@tothenew.com", yearOfExperience: 2.5, isPresent: true,
                                    competency: "AI", attendancePercent: 92),
                                employee(empID: 2, empName: "Rhythm", empEmail: "rhythm@tothenew.com", yearOfExperience: 0.8, isPresent: true,
                                    competency: "android", attendancePercent: 88),
                                employee(empID: 5, empName: "Harsh", empEmail: "harsh@tothenew.com", yearOfExperience: 4.0, isPresent: false,
                                    competency: "android", attendancePercent: 90),
                                employee(empID: 6, empName: "Vijender", empEmail: "vijender@tothenew.com", yearOfExperience: 1.5, isPresent:
                                    true, competency: "BigData", attendancePercent: 76),
                                employee(empID: 7, empName: "Mithilesh", empEmail: "mithilesh@tothenew.com", yearOfExperience: 1.1, isPresent:
                                    true, competency: "BigData", attendancePercent: 81),
                                employee(empID: 8, empName: "Merry", empEmail: "merry@tothenew.com", yearOfExperience: 1.5, isPresent: true,
                                    competency: "iOS", attendancePercent: 90),
                                employee(empID: 9, empName: "Aniket", empEmail: "aniket@tothenew.com", yearOfExperience: 2.0, isPresent: true,
                                    competency: "QE", attendancePercent: 90),
                                employee(empID: 10, empName: "Sachin", empEmail: "sachin@tothenew.com", yearOfExperience: 3.5, isPresent:
                                    true, competency: "AI", attendancePercent: 85)]



```



```swift
enum bonusError: Error{
    case isPresent (String)
    case yeo (String)
    case nothotCompetency (String)
    case attendance (String)
    var localizedDescription: String{
        switch self {
        case .isPresent(let name):
            return name + "is absent today"
        case .yeo(let name):
            return name + "is still to complete an year with us"
        case .nothotCompetency(let name):
            return name + "competency does not fall under bonus program."
        case .attendance(let name):
            return name + "has attendance percent less than 80"
        }
    }
}



func allowedForBonus(bonusdata : [employee]) throws -> Void {
    for item in employeeData{
        if item.isPresent == true{
            if item.yearOfExperience > 1.0{
                if item.competency == "iOS" || item.competency == "android" || item.competency == "BigData" || item.competency == "AI"{
                    if item.attendancePercent > 80{
                        print ("\(item.empName) is eligible for bonus")
                    }else{
                        throw bonusError.attendance(item.empName)
                    }
                }else{
                    throw bonusError.nothotCompetency(item.empName)
                }
```



```swift
            }
            else{
                throw bonusError.yeo(item.empName)
            }
        }
        else{
            throw bonusError.isPresent(item.empName)
        }
    }
//    return nil
}

//let result = try allowedForBonus(bonusdata: employeeData)

do {
    let _ = try allowedForBonus(bonusdata: employeeData)
} catch bonusError.isPresent(let name){
    print(name + " is absent today")
} catch bonusError.nothotCompetency(let name){
    print(name + " competancy does not fall under bonus program")
} catch bonusError.yeo(let name){
    print(name + " is still to complete a year with us")
} catch bonusError.attendance(let name) {
    print(name + " has attendance less than 80")
}
```

```
Aryan is eligible for bonus
Kavya is eligible for bonus
Rahul is eligible for bonus
Rhythm is still to complete a year with us
```