1. What is an extension?

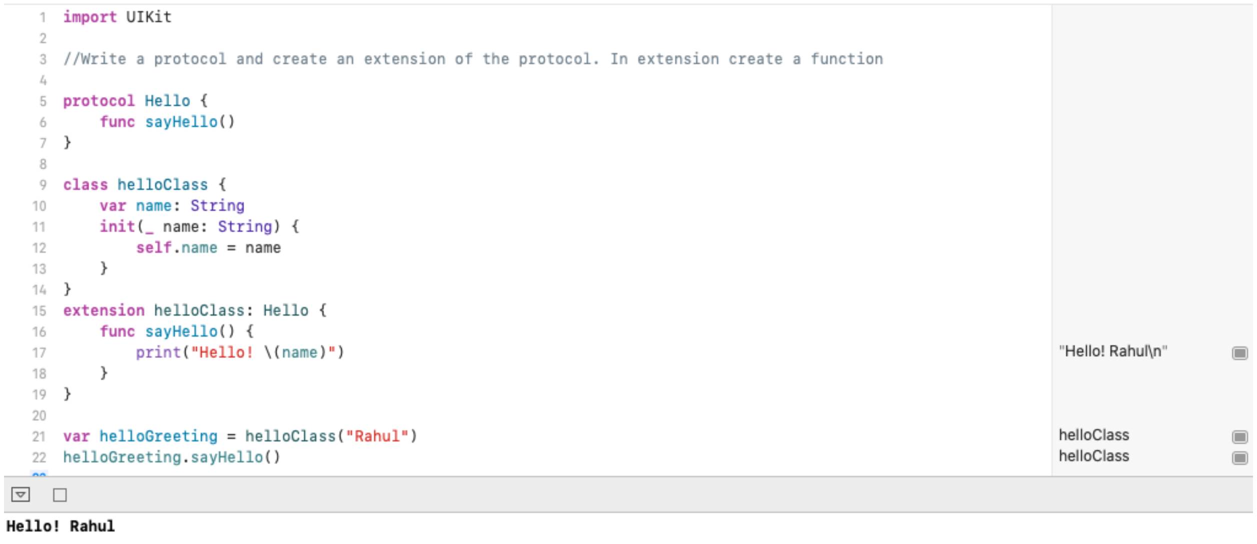
Ans - Swift Extension is a useful feature that helps in adding more functionality to an existing Class, Structure, Enumeration or a Protocol type. This includes adding functionalities for types where you don’t have the original source code too (extensions for Int, Bool, String etc. types). For example(extension for Double class) :

```
extension Double {

   var km: Double { return self * 1_000.0 }

   var m: Double { return self }

   var cm: Double { return self / 100.0 }

   var mm: Double { return self / 1_000.0 }

   var ft: Double { return self / 3.28084 }

      }
```

2. Create a class and write the delegate of UITextField in extension of that class.

Ans - Delegates were not covered in this session in depth.

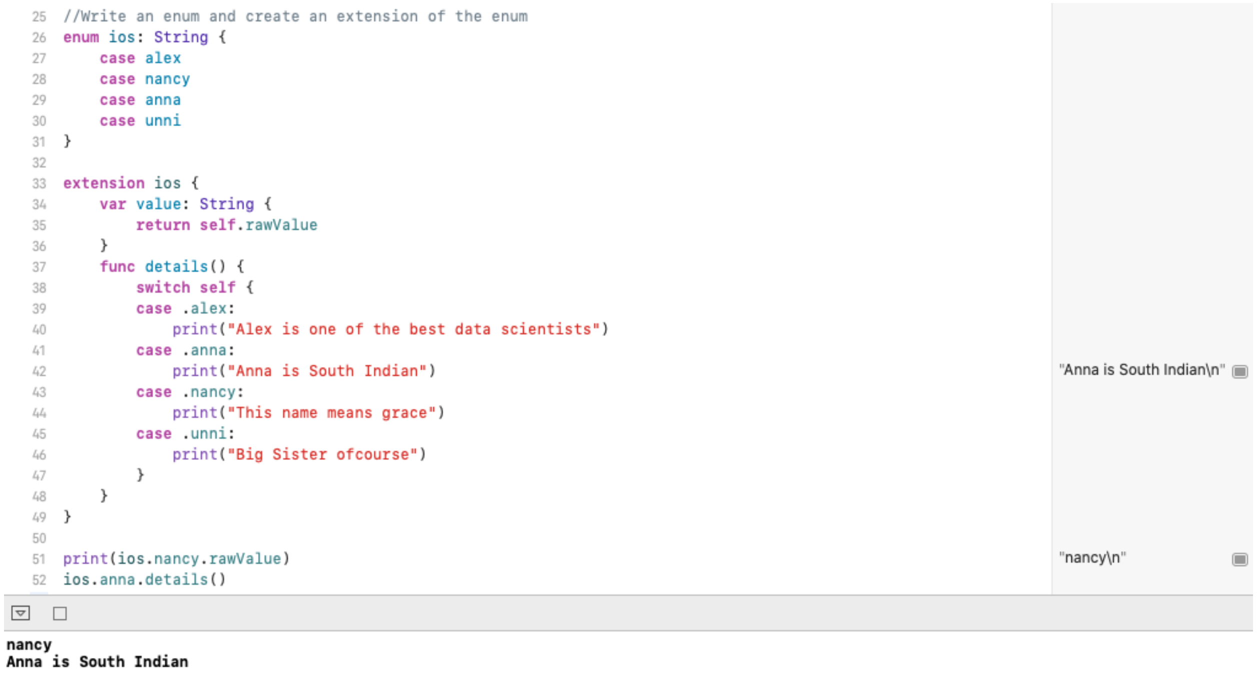
3. Write a protocol and create an extension of the protocol. In extension create a function



```swift
import UIKit

//Write a protocol and create an extension of the protocol. In extension create a function

protocol Hello {
    func sayHello()
}

class helloClass {
    var name: String
    init(_ name: String) {
        self.name = name
    }
}
extension helloClass: Hello {
    func sayHello() {
        print("Hello! \(name)")                         "Hello! Rahul\n"
    }
}

var helloGreeting = helloClass("Rahul")                 helloClass
helloGreeting.sayHello()                                helloClass
```

```
Hello! Rahul
```

4.Write an enum and create an extension of the enum.



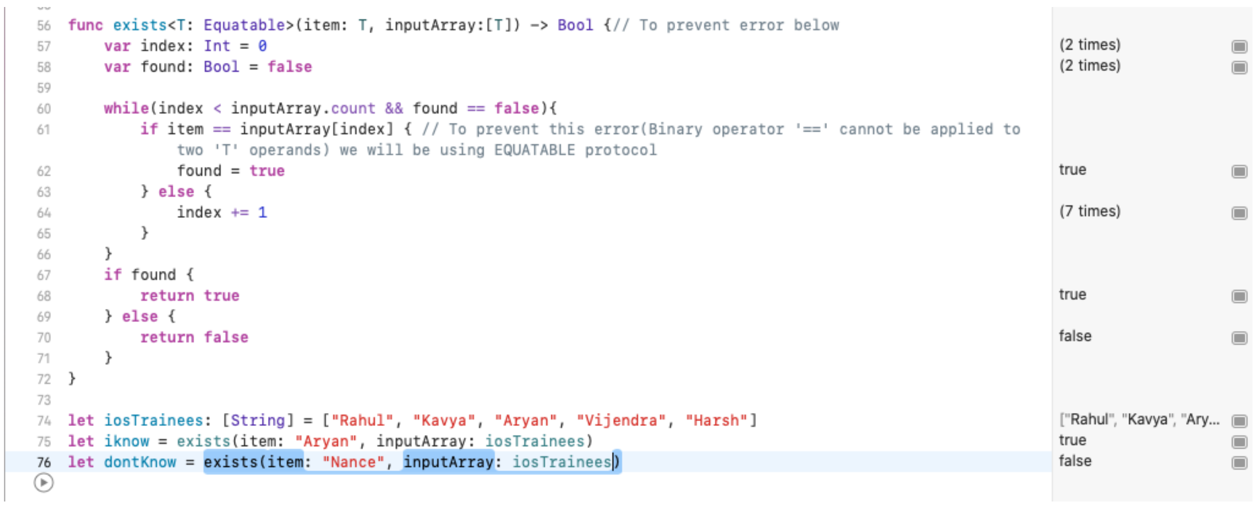
```swift
//Write an enum and create an extension of the enum
enum ios: String {
    case alex
    case nancy
    case anna
    case unni
}

extension ios {
    var value: String {
        return self.rawValue
    }
    func details() {
        switch self {
        case .alex:
            print("Alex is one of the best data scientists")
        case .anna:
            print("Anna is South Indian")                "Anna is South Indian\n"
        case .nancy:
            print("This name means grace")
        case .unni:
            print("Big Sister ofcourse")
        }
    }
}

print(ios.nancy.rawValue)                                "nancy\n"
ios.anna.details()
```

```
nancy
Anna is South Indian
```

5. What is Generic?

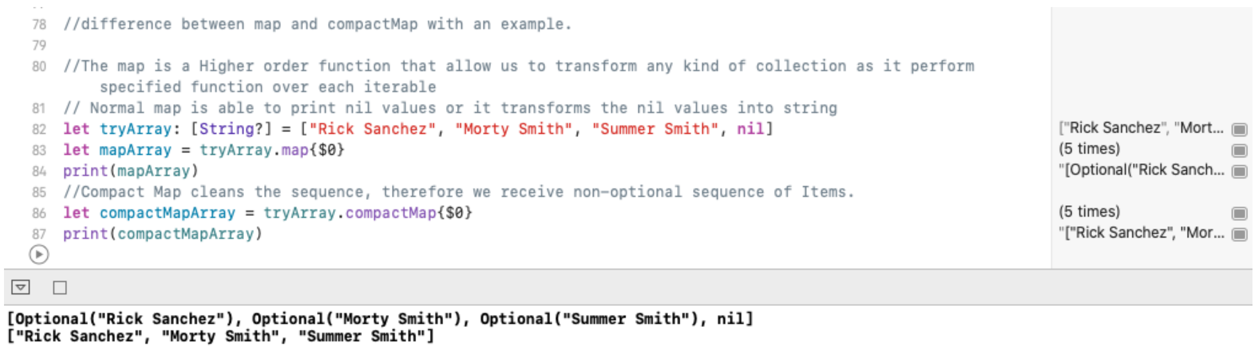
Generic codes enables us to write flexile, reusable functions without any specific bound data types for the requirements that we provide. We can write code that avoids duplication and expresses its intent in a clear, abstracted manner.

6. Explain generic with an example?



```swift
func exists<T: Equatable>(item: T, inputArray:[T]) -> Bool {// To prevent error below
    var index: Int = 0
    var found: Bool = false

    while(index < inputArray.count && found == false){
        if item == inputArray[index] { // To prevent this error(Binary operator '==' cannot be applied to
            two 'T' operands) we will be using EQUATABLE protocol
            found = true
        } else {
            index += 1
        }
    }
    if found {
        return true
    } else {
        return false
    }
}

let iosTrainees: [String] = ["Rahul", "Kavya", "Aryan", "Vijendra", "Harsh"]
let iknow = exists(item: "Aryan", inputArray: iosTrainees)
let dontKnow = exists(item: "Nance", inputArray: iosTrainees)
```

7. Explain the difference between map and compactMap with an example.

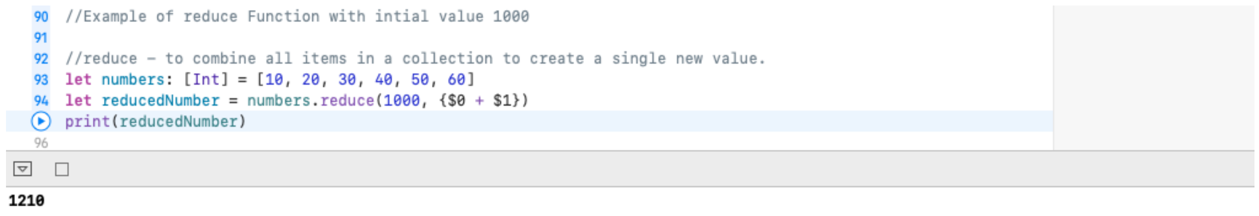


```swift
//difference between map and compactMap with an example.

//The map is a Higher order function that allow us to transform any kind of collection as it perform
    specified function over each iterable
// Normal map is able to print nil values or it transforms the nil values into string
let tryArray: [String?] = ["Rick Sanchez", "Morty Smith", "Summer Smith", nil]
let mapArray = tryArray.map{$0}
print(mapArray)
//Compact Map cleans the sequence, therefore we receive non-optional sequence of Items.
let compactMapArray = tryArray.compactMap{$0}
print(compactMapArray)
```

["Rick Sanchez", "Mort...
(5 times)
"[Optional("Rick Sanch...
(5 times)
"["Rick Sanchez", "Mor...

```
[Optional("Rick Sanchez"), Optional("Morty Smith"), Optional("Summer Smith"), nil]
["Rick Sanchez", "Morty Smith", "Summer Smith"]
```

8. Write an example of a reduced function with initial value 1000.

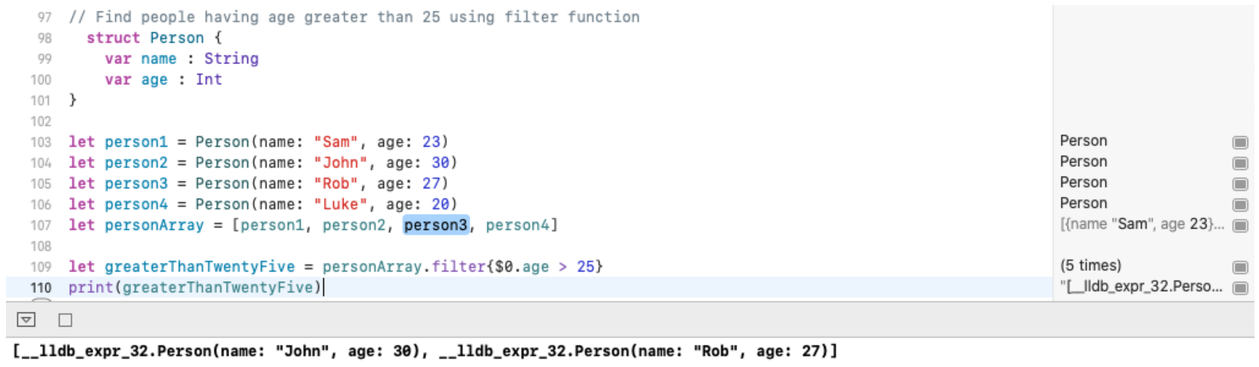


```swift
//Example of reduce Function with intial value 1000

//reduce - to combine all items in a collection to create a single new value.
let numbers: [Int] = [10, 20, 30, 40, 50, 60]
let reducedNumber = numbers.reduce(1000, {$0 + $1})
print(reducedNumber)
```

```
1210
```

9. - 2 marks

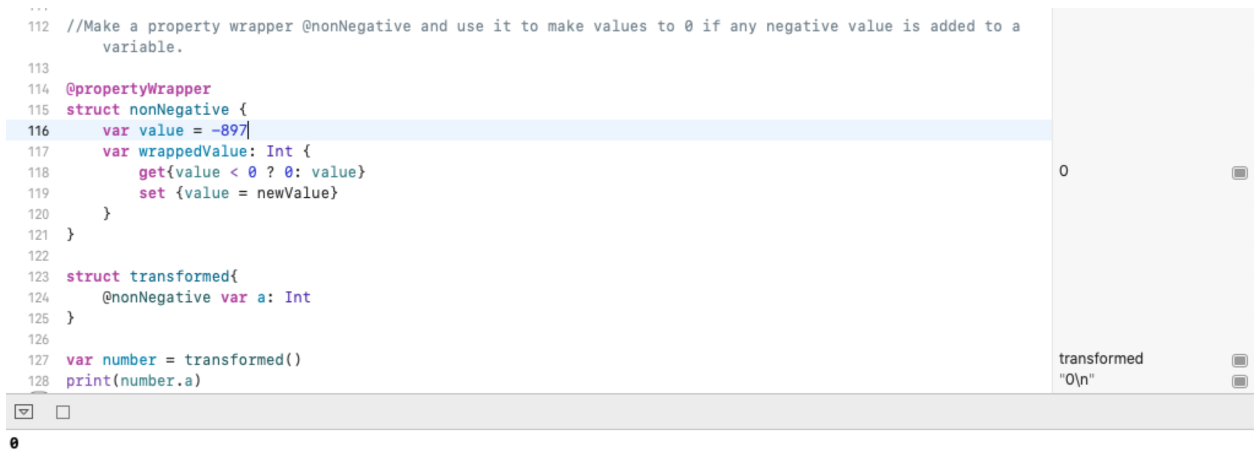
Find all people whose age is more than 25 using a filter function.



```swift
// Find people having age greater than 25 using filter function
  struct Person {
    var name : String
    var age : Int
}

let person1 = Person(name: "Sam", age: 23)
let person2 = Person(name: "John", age: 30)
let person3 = Person(name: "Rob", age: 27)
let person4 = Person(name: "Luke", age: 20)
let personArray = [person1, person2, person3, person4]

let greaterThanTwentyFive = personArray.filter{$0.age > 25}
print(greaterThanTwentyFive)
```

```
[__lldb_expr_32.Person(name: "John", age: 30), __lldb_expr_32.Person(name: "Rob", age: 27)]
```

10. Make a property wrapper @nonNegative and use it to make values to 0 if any negative value is added to a variable.



```swift
//Make a property wrapper @nonNegative and use it to make values to 0 if any negative value is added to a
    variable.

@propertyWrapper
struct nonNegative {
    var value = -897
    var wrappedValue: Int {
        get{value < 0 ? 0: value}
        set {value = newValue}
    }
}

struct transformed{
    @nonNegative var a: Int
}

var number = transformed()
print(number.a)
```

```
0
```