Date: 19.02.2021

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

SWIFT ADVANCE Assignment

1. What is extension?

   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).

   Following are the essential functionalities that Swift Extension offer us:

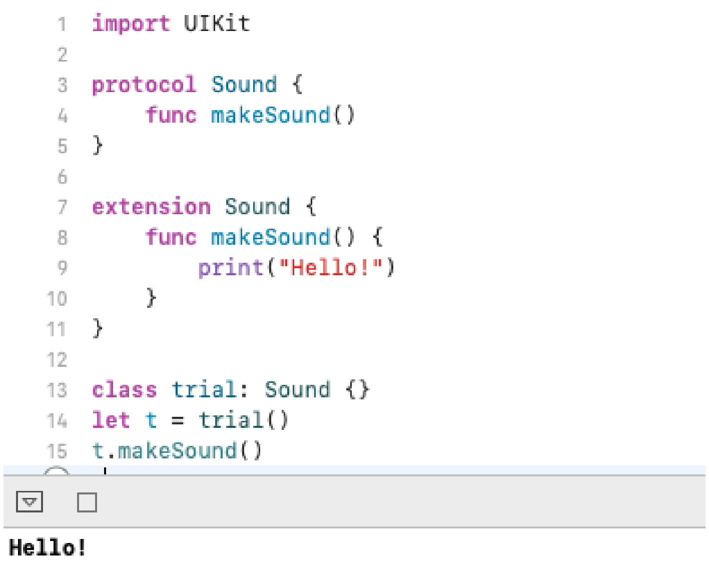
   - Add computed instance properties and computed type properties
   - Define instance methods and type methods
   - Provide new initializers
   - Define subscripts
   - Define and use new nested types
   - Make an existing type conform to a protocol

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

We have not yet covered this topic in the session.

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

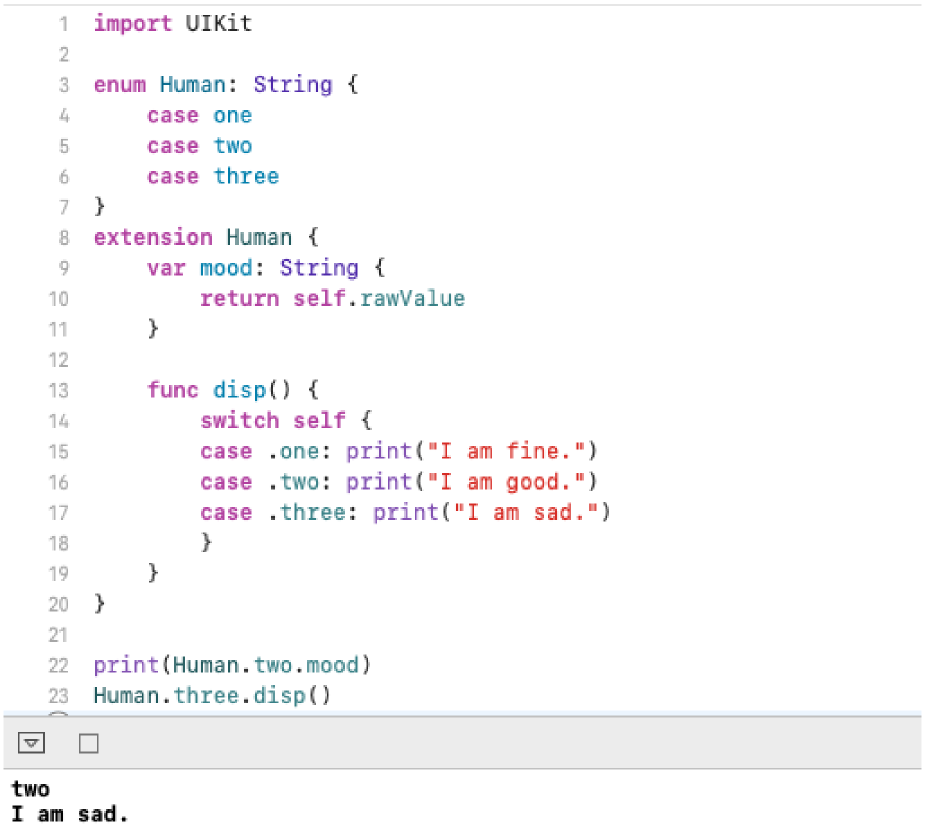
```
    func sayHello() {

     print("Hello!")

}
```



```swift
import UIKit

protocol Sound {
    func makeSound()
}

extension Sound {
    func makeSound() {
        print("Hello!")
    }
}

class trial: Sound {}
let t = trial()
t.makeSound()
```

```
Hello!
```

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



```swift
import UIKit

enum Human: String {
    case one
    case two
    case three
}
extension Human {
    var mood: String {
        return self.rawValue
    }

    func disp() {
        switch self {
        case .one: print("I am fine.")
        case .two: print("I am good.")
        case .three: print("I am sad.")
        }
    }
}

print(Human.two.mood)
Human.three.disp()
```

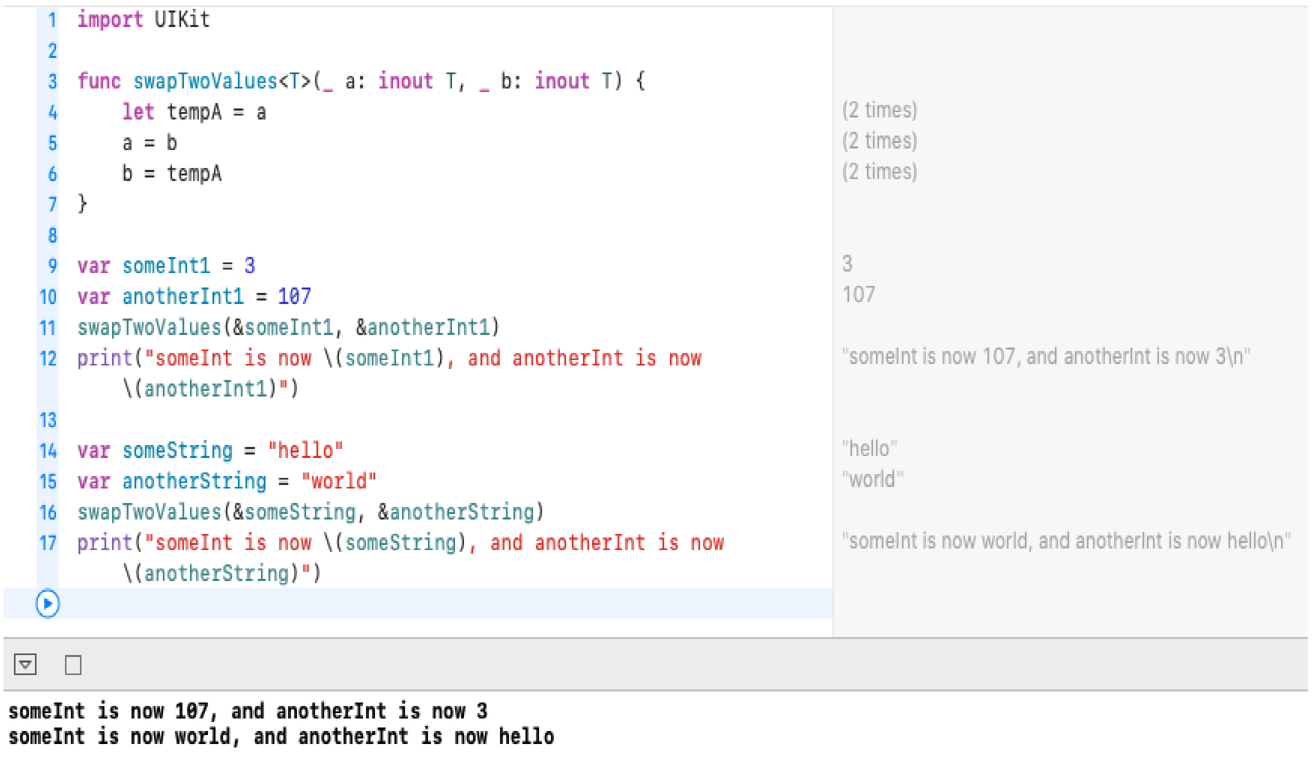
```
two
I am sad.
```

5. What is Generic?

Generic code enables you to write flexible, reusable functions and types that can work with any type, subject to requirements that you define. You can write code that avoids duplication and expresses its intent in a clear, abstracted manner. Generics are one of the most powerful features of Swift, and much of the Swift standard library is built with generic code.

Swift's Array and Dictionary types are both generic collections. You can create an array that holds Int values, or an array that holds String values, or indeed an array for any other type that can be created in Swift. Similarly, you can create a dictionary to store values of any specified type, and there are no limitations on what that type can be.

6. Explain generic with an example?



```swift
import UIKit

func swapTwoValues<T>(_ a: inout T, _ b: inout T) {
    let tempA = a
    a = b
    b = tempA
}

var someInt1 = 3
var anotherInt1 = 107
swapTwoValues(&someInt1, &anotherInt1)
print("someInt is now \(someInt1), and anotherInt is now
    \(anotherInt1)")

var someString = "hello"
var anotherString = "world"
swapTwoValues(&someString, &anotherString)
print("someInt is now \(someString), and anotherInt is now
    \(anotherString)")
```

```
(2 times)
(2 times)
(2 times)

3
107
"someInt is now 107, and anotherInt is now 3\n"

"hello"
"world"

"someInt is now world, and anotherInt is now hello\n"
```

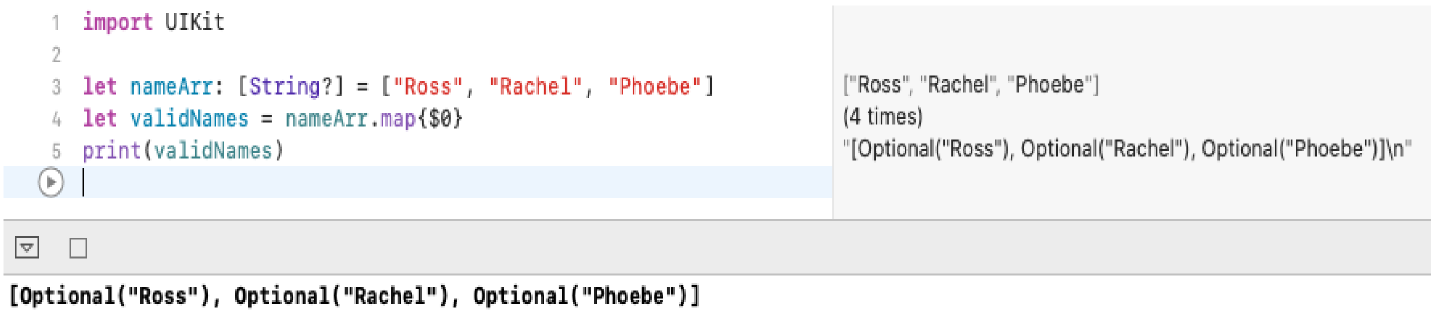
```
someInt is now 107, and anotherInt is now 3
someInt is now world, and anotherInt is now hello
```

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

Map: This higher order function loops over a collection and applies the same operation to each element in the collection.

let resultCollection = inputCollection.map { (elementOfCollection) - ResultType in
    return //something related to the element
  }

EXAMPLE:



```swift
import UIKit

let nameArr: [String?] = ["Ross", "Rachel", "Phoebe"]
let validNames = nameArr.map{$0}
print(validNames)
```

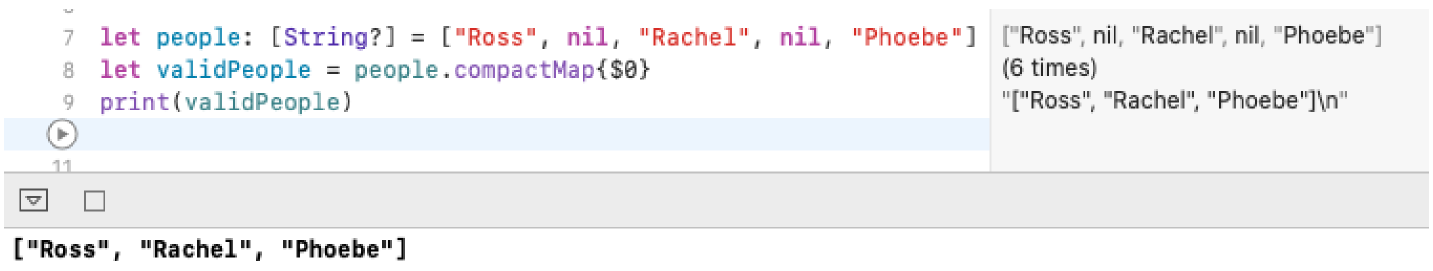
```
["Ross", "Rachel", "Phoebe"]
(4 times)
"[Optional("Ross"), Optional("Rachel"), Optional("Phoebe")]\n"
```

```
[Optional("Ross"), Optional("Rachel"), Optional("Phoebe")]
```

compactMap: CompactMap is same as the Map function with optional handling capability. CompactMap is also used to filter out the nil value.

```
let resultCollection = inputCollection.compactMap {
(elementOfCollection) ->  ResultType in
   return //something related to the element
}
```

EXAMPLE:

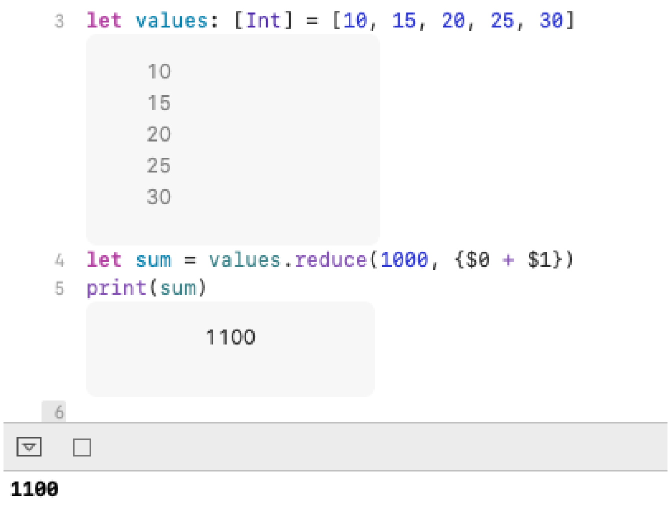


```swift
let people: [String?] = ["Ross", nil, "Rachel", nil, "Phoebe"]
let validPeople = people.compactMap{$0}
print(validPeople)
```

```
["Ross", nil, "Rachel", nil, "Phoebe"]
(6 times)
"["Ross", "Rachel", "Phoebe"]\n"
```

```
["Ross", "Rachel", "Phoebe"]
```

8. Write an example of reduce function with initial value 1000.



```
let values: [Int] = [10, 15, 20, 25, 30]
    10
    15
    20
    25
    30
let sum = values.reduce(1000, {$0 + $1})
print(sum)
    1100

1100
```

9. struct Person {

    var name : String

    var age : Int

    }

    let person1 = Person(name: "Sam", age: 23)

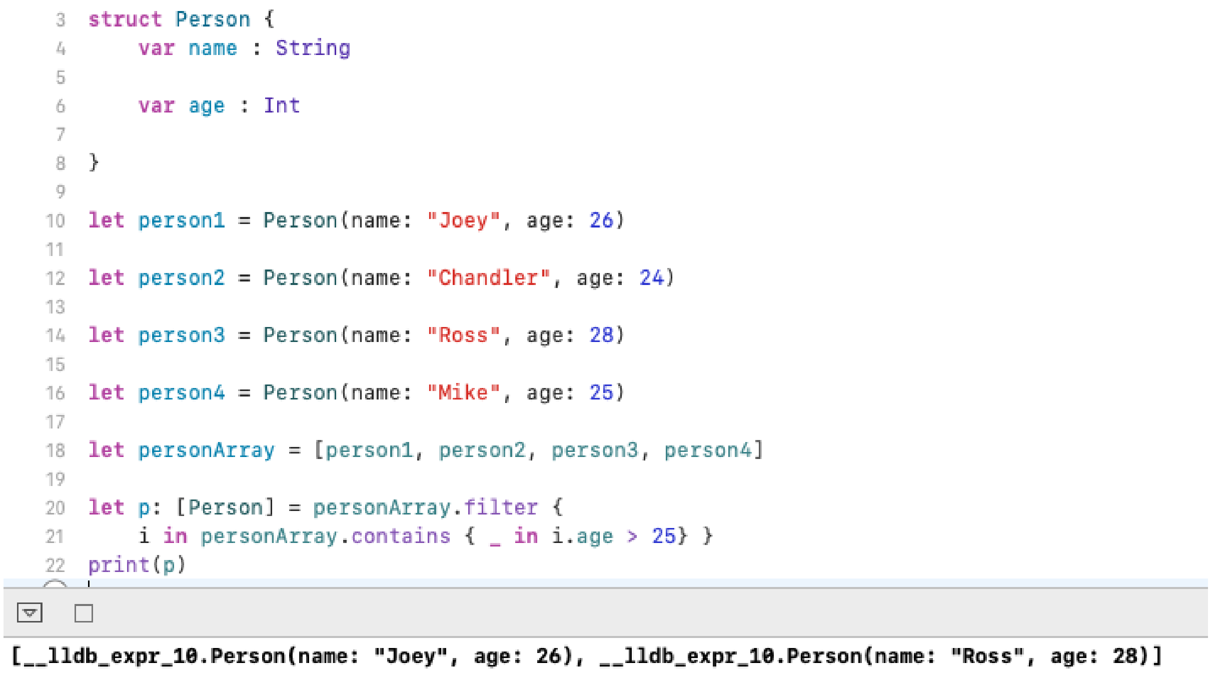
    let person2 = Person(name: "John", age: 30)

    let person3 = Person(name: "Rob", age: 27)

    let person4 = Person(name: "Luke", age: 20)

    let personArray = [person1, person2, person3, person4]

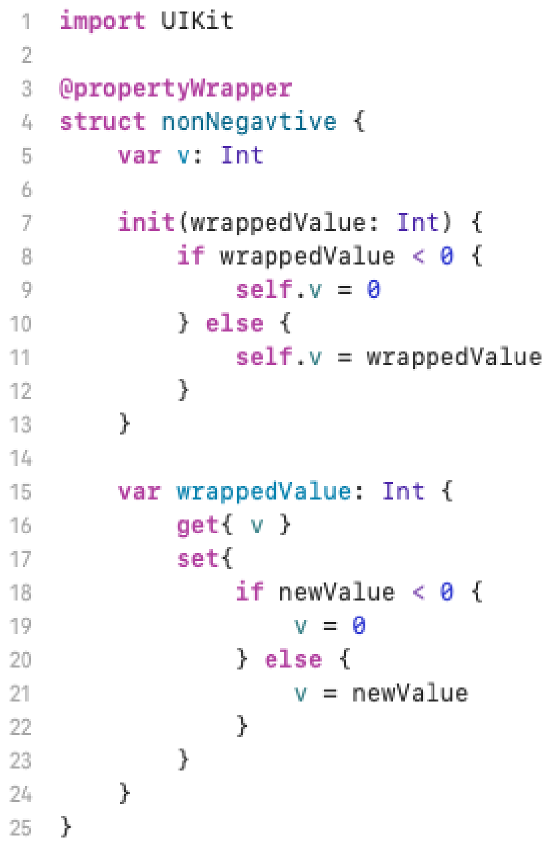
    Find all person whose age is more than 25 using filter function.



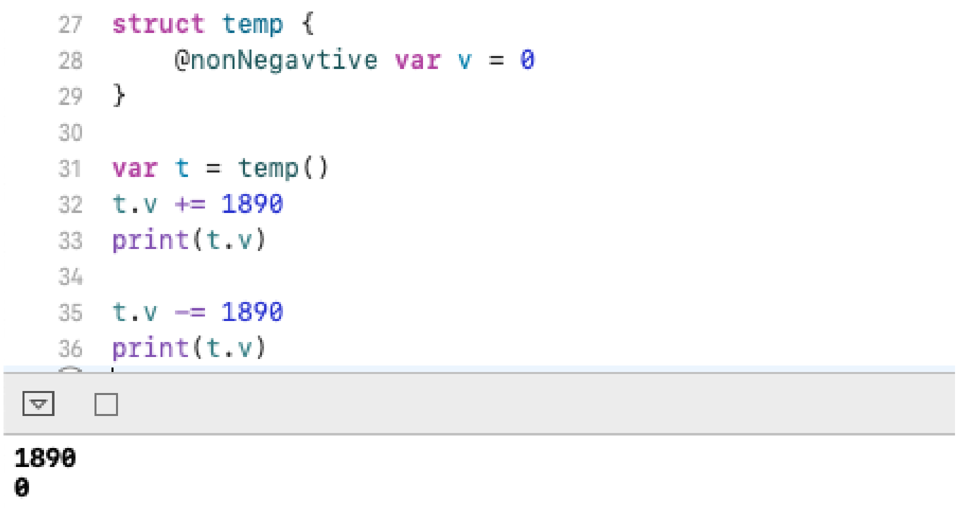
```swift
struct Person {
    var name : String

    var age : Int

}

let person1 = Person(name: "Joey", age: 26)

let person2 = Person(name: "Chandler", age: 24)

let person3 = Person(name: "Ross", age: 28)

let person4 = Person(name: "Mike", age: 25)

let personArray = [person1, person2, person3, person4]

let p: [Person] = personArray.filter {
    i in personArray.contains { _ in i.age > 25} }
print(p)
```

```
[__lldb_expr_10.Person(name: "Joey", age: 26), __lldb_expr_10.Person(name: "Ross", age: 28)]
```

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



```swift
import UIKit

@propertyWrapper
struct nonNegavtive {
    var v: Int

    init(wrappedValue: Int) {
        if wrappedValue < 0 {
            self.v = 0
        } else {
            self.v = wrappedValue
        }
    }

    var wrappedValue: Int {
        get{ v }
        set{
            if newValue < 0 {
                v = 0
            } else {
                v = newValue
            }
        }
    }
}
```



```swift
struct temp {
    @nonNegavtive var v = 0
}

var t = temp()
t.v += 1890
print(t.v)

t.v -= 1890
print(t.v)
```

```
1890
0
```