Swift Advance

Exercise 5

Abhishek Maurya

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).
2. **Create a class and write the delegate of UITextField in extension of that class.**
    - `extension UIViewController : UITextFieldDelegate {}`
3. **Write a protocol and create an extension of the protocol. In extension create a function**

    func sayHello() {

    print("Hello!")

    }

    ```
    protocol sayhelloP {
        func sayHello()
    }
    extension sayHey : sayhelloP {
        func sayHello() {
            print("Hello!")
        }
    }
    class sayHey {
        var name: String?
        }
    var ans3 = sayHey()
    ans3.sayHello()
    ```

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

    ```
    enum myEnum1: String {
        case enum1
        case enum2
        case enum3
    }
    extension myEnum1 {
        var value: String {
            return self.rawValue
        }
        func printValue() {
            switch self {
            case .enum1:
                print("How are you")
            case .enum2:
                print("How was your day")
            case .enum3:
                print("How are you doing")
            }
        }
    ```

```
}
myEnum1.enum2.printValue()
```

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.
6. **Explain generic with an example?**
   - ```
     //non-generic addition function for int

     func addition1(a: Int, b: Int) -> Int {
         return a + b
     }
     print(addition1(a: 3, b: 4))

     //non-generic addition function for double
     func addition2(a: Double, b: Double) -> Double {
         return a + b
     }
     print(addition2(a: 3, b: 4))

     //generic function
     func addition3<T: Numeric>(a: T, b: T) -> T{
         return a + b
     }
     print(addition3(a: 3, b: 4))
     ```
7. **Explain the difference between map and compactMap with an example.**
   - Map : Map is a high order function which Returns an array containing the results of mapping the given closure over the sequence's elements.
   - CompactMap : CompactMap is a high order function which Returns an array containing the non-nil results of calling the given transformation with each element of this sequence.

     ```
     let anArray = [1,nil,2,nil,3,4]
     let Maparray = anArray.map{$0}
     print(Maparray)
     let MapCompactArray = anArray.compactMap{$0}
     print(MapCompactArray)
     ```
8. **Write an example of reduce function with initial value 1000.**
   - ```
     var maxNumber = Array(1...5)
     .reduce(1000) { (total, number) in (total + number) }
     print(maxNumber)
     ```
9. **2 marks**

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.

- ```
  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]
  var moreThentf = personArray.filter{$0.age > 25}
  for emp1 in moreThentf {
      print(emp1.name," is older then 25")
  }
  ```
  ```