iOS: Swift Basics 2

Exercise 3

Abhishek Maurya

- *Initializers*

1. **Implement the parameterised initialisation with class or struct.**

    - ```
      class myClass1 {
          var entity1: String?
          var entity2: String?
          var entity3: Int?
          init(e1: String, e2: String, e3: Int) {
              self.entity1 = e1
              self.entity2 = e2
              self.entity3 = e3
          }
      }
      ```

2. **Write all the Rules of initialiser in Inheritance**

    - A designated initializer must call a designated initializer from its immediate superclass.

    - A convenience initializer must call another initializer from the same class.

    - A convenience initializer must ultimately call a designated initializer.

3. **Using convenience Initializers, write-down the Initializers for MOVIE class having basic**

    - ```
      class MOVIE {
          var title: String?
          var author: String?
          var publish_date: String?
          init(movieName : String , producer : String , date : String)
          {
              self.title = movieName
              self.author = producer
              self.publish_date = date
          }
          convenience init( title : String) {
              self.init(movieName: title, producer: "Richard", date:
      "3rd Oct")
          }
      }
      ```

4. **attributes like title, author, publish_date, etc.**

    - ```
      // calling convinince initializer
      var movie1 = MOVIE(title: "John Wich")
      print("\(movie1.title!), \(movie1.author!),
      \(movie1.publish_date!)")

      //calling designated Initialiser
      var movie2 = MOVIE(movieName: "Deadpool", producer: "Marvels",
      date: "6th Oct")
      print("\(movie2.title!), \(movie2.author!),
      \(movie2.publish_date!)")
      ```

5. **Declare a structure which can demonstrate the throwable Initializer**

```
● struct MyTestStruct {
      var text: String
      init() throws {
          text = try String(/*some code here*/)
      }
  }
```

● *Array*

1. **Create an array containing the 5 different integer values. Write are at least 4 ways to do this.**

```
● var array1 = Array<String>() //Type 1
  var array2: Array<String> = [] //Type 2
  var array3 = [String]() //Type 3
  var array4: [String] = [] //Type 4
```

2. **Create an immutable array containing 5 city names.**

```
● let cities = ["New Delhi", "Kolkata", "Mumbai", "Chennai",
  "Banglore"]
```

3. **Create an array with city 5 city names. Later add other names like Canada, Switzerland, Spain to the end of the array in at least 2 possible ways.**

```
● var cities1 = ["Sidney", "Kolkata", "Mumbai", "Chennai",
  "Banglore"]
  // First method
  cities1.append("Canada")
  cities1.append("Switzerland")
  //Second Method
  cities1 += ["Spain", "Germany"]
```

4. **Create an array with values 14, 18, 15, 16, 23, 52, 95. Replace the values 24 & 48 at 2nd & 4th index of array**

```
● var array = [14, 18, 15, 16, 23, 52, 95]
  array[2] = 24
  array[4] = 48
```

● *Set*

Given the following sets:

let houseAnimals: Set = [" 🐶 ", " 🐱 "]

let farmAnimals: Set = [" 🐮 ", " 🐔 ", " 🐑 ", " 🐶 ", " 🐱 "]

let cityAnimals: Set = [" 🐦 ", " 🐭 "]

Use set operations to...

1. **Determine whether the set of house animals is a subset of farm animals.**

- if houseAnimals.isSubset(of: farmAnimals) {
        print("True")
    }

2. **Determine whether the set of farm animals is a superset of house animals.**
    - if houseAnimals.isSuperset(of: houseAnimals) {
          print("True")
      }

3. **Determine if the set of farm animals is disjoint with city animals.**
    - if houseAnimals.isDisjoint(with: cityAnimals) {
          print("True")
      }

4. **Create a set that only contains farm animals that are not also house animals.**
    - let ans3_4 = farmAnimals.subtracting(houseAnimals)

5. **Create a set that contains all the animals from all sets.**
    - let ans3_5 = farmAnimals.union(cityAnimals)

- *Dictionary*

    1. **Create an empty dictionary with keys of type String and values of type Int and assign it to a variable in as many ways as you can think of (there's at least 4 ways).**
        - ```
          let dictionary1: Dictionary<String, Int> = [:] //Type 1
          let dictionary2: [String: Int] = [:] //Type 2
          let dictionary3 = Dictionary<String, Int>() //Type 3
          let dictionary4 = [String: Int]() //Type 4
          ```

    2. **Create a mutable dictionary named secretIdentities where the key value pairs are "Hulk" -> "Bruce Banner", "Batman" -> "Bruce Wayne", and "Superman" -> "Clark Kent".**
        - var secretIdentities: [String:String] = ["Hulk":"Bruce Banner","Batman":"Bruce Wayne","Superman":"Clark Kent"]

    3. **Create a nesters structure of Key-value pair.**
        - var nestedDictionary: [Int:[Int:String]]=[1:[1:"Akash",2:"aditya"],2:[1:"anubhav",2:"shivam"]]

    4. **Print all the keys in the dictionary**
        - print(nestedDictionary.keys)

- *Subscript*

    1. **What is subscript? Write down the declaration syntax.**
        - Classes, structures, and enumerations can define subscripts, which are shortcuts for accessing the member elements of a collection, list, or sequence.
        - subscript(index: Int) -> Int {
              //return code
          }

    2. **Create a simple subscript that outputs true if a string contains a substring and false otherwise.**
        - class daysofaweek {
              private var days = ["Sunday", "Monday", "Tuesday",

```
        "Wednesday", "Thursday", "Friday", "Saturday"]
            subscript(index: Int) -> Bool {
                get {
                    if days[index].contains("Sunday"){
                        return true
                    }
                    else{
                        return false
                    }
                }
            }
        }
```