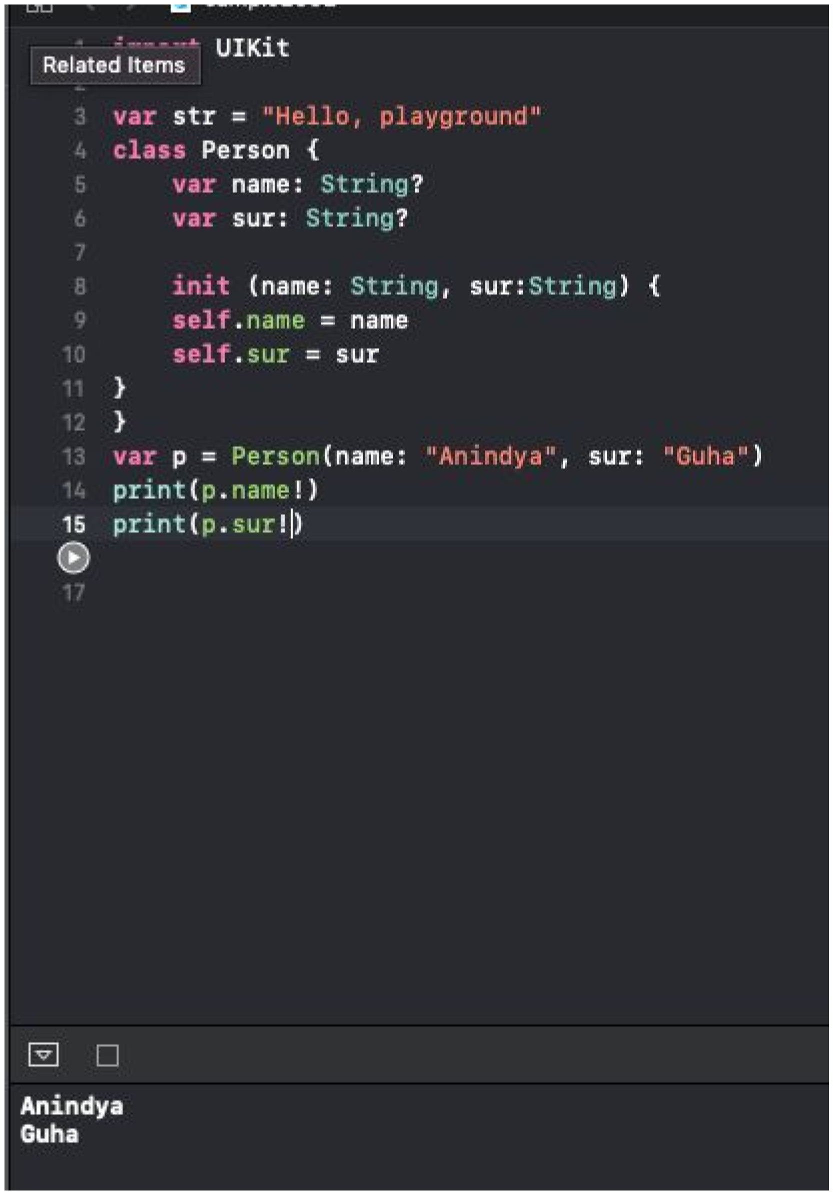
Exercise :
// Initializers

1. Implement the parameterised initialisation with class or struct.



```swift
import UIKit

var str = "Hello, playground"
class Person {
    var name: String?
    var sur: String?

    init (name: String, sur:String) {
    self.name = name
    self.sur = sur
}
}
var p = Person(name: "Anindya", sur: "Guha")
print(p.name!)
print(p.sur!)
```

```
Anindya
Guha
```

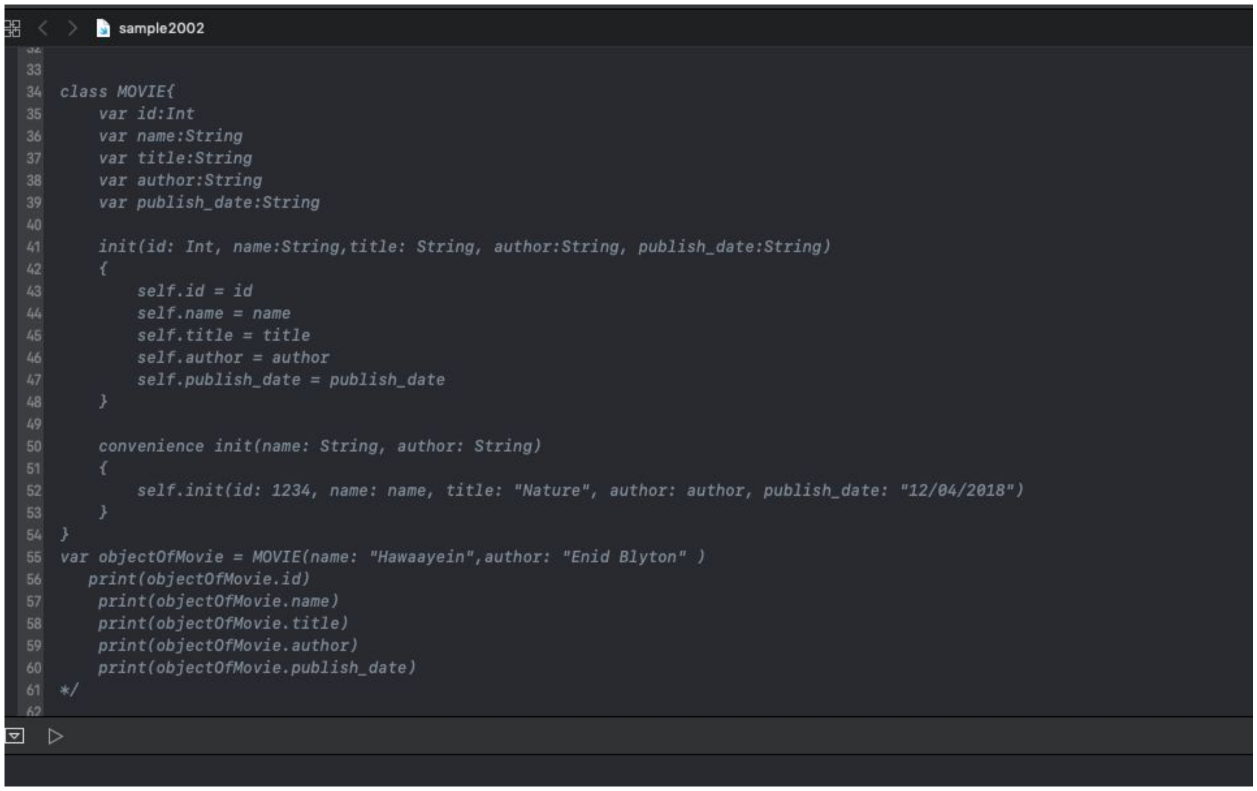
2. Write all the Rules of initialiser in Inheritance

Rule 1: A designated initializer must call a designated initializer from its immediate superclass.

Rule 2: A convenience initializer must call another initializer from the same class.

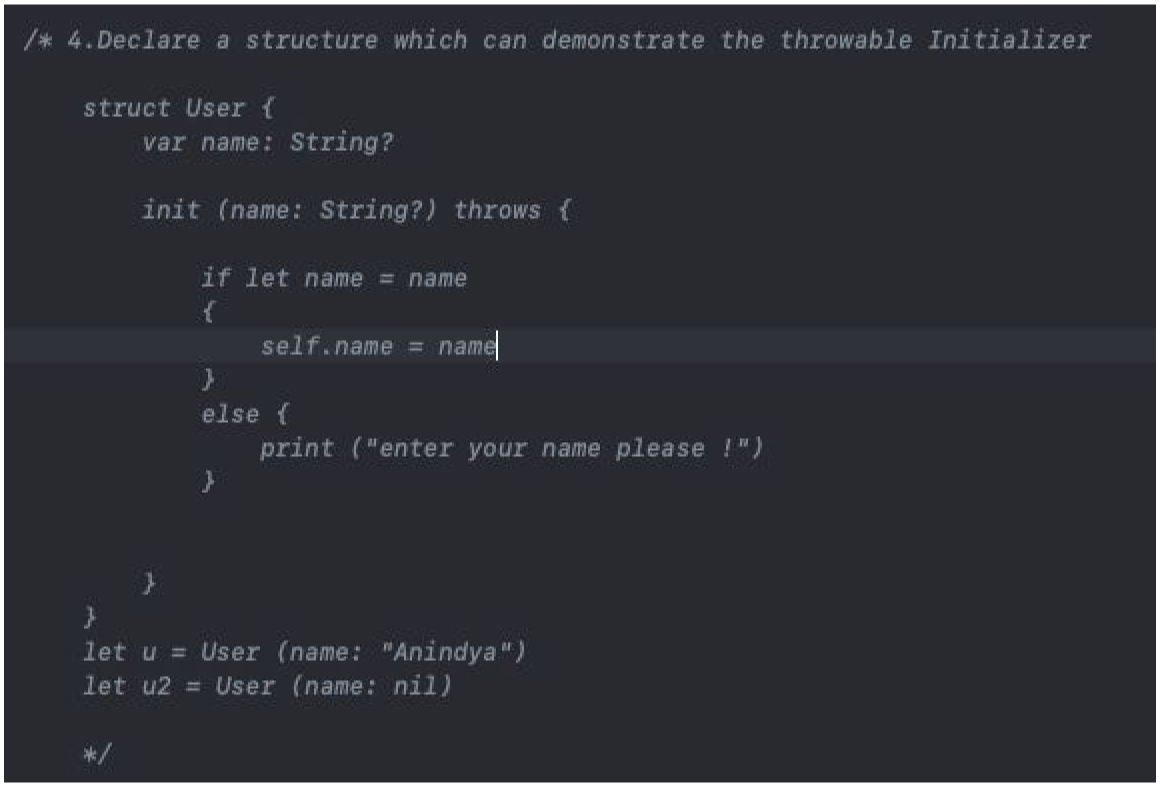
Rule 3: A convenience initializer must ultimately call a designated initializer

3. Using convenience **Initializers**, write-down the **Initializers** for MOVIE class having basic attributes like title, author, publish_date, etc.



```
class MOVIE{
    var id:Int
    var name:String
    var title:String
    var author:String
    var publish_date:String

    init(id: Int, name:String,title: String, author:String, publish_date:String)
    {
        self.id = id
        self.name = name
        self.title = title
        self.author = author
        self.publish_date = publish_date
    }

    convenience init(name: String, author: String)
    {
        self.init(id: 1234, name: name, title: "Nature", author: author, publish_date: "12/04/2018")
    }
}
var objectOfMovie = MOVIE(name: "Hawaayein",author: "Enid Blyton" )
   print(objectOfMovie.id)
    print(objectOfMovie.name)
    print(objectOfMovie.title)
    print(objectOfMovie.author)
    print(objectOfMovie.publish_date)
*/
```

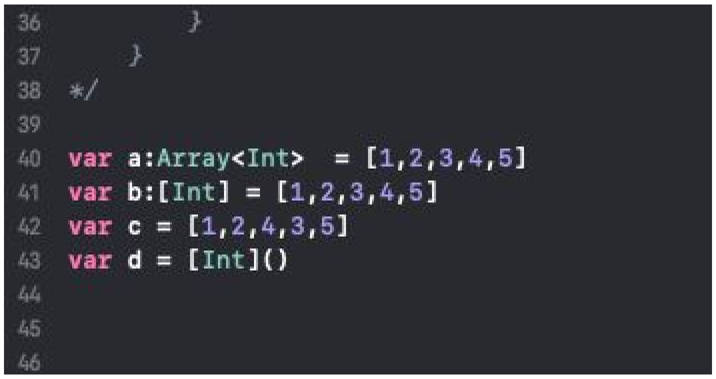
4. Declare a structure which can demonstrate the throwable Initializer



```
/* 4.Declare a structure which can demonstrate the throwable Initializer

    struct User {
        var name: String?

        init (name: String?) throws {

            if let name = name
            {
                self.name = name
            }
            else {
                print ("enter your name please !")
            }


        }
    }
    let u = User (name: "Anindya")
    let u2 = User (name: nil)

    */
```

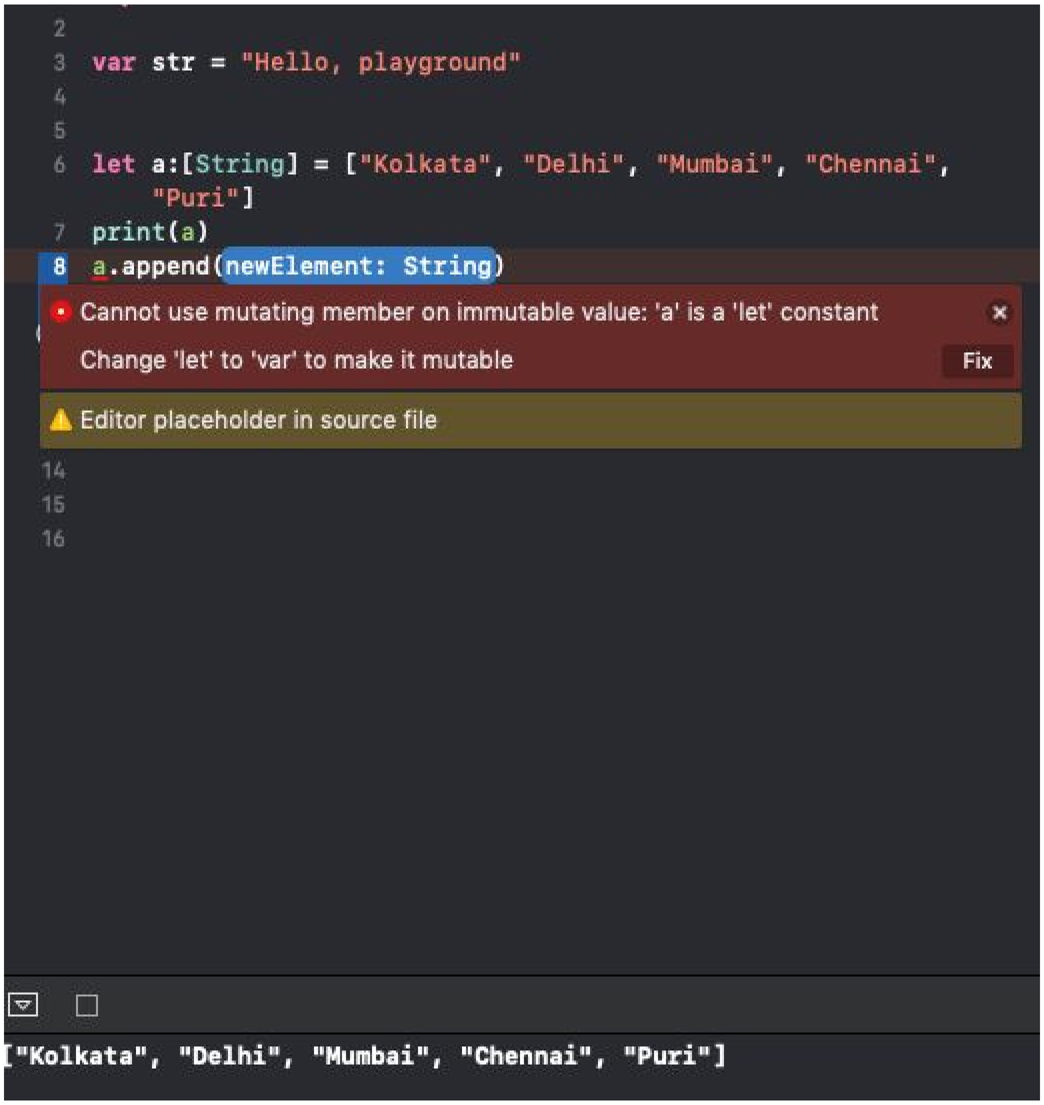
// Array

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



```
            }
        }
*/

var a:Array<Int>  = [1,2,3,4,5]
var b:[Int] = [1,2,3,4,5]
var c = [1,2,4,3,5]
var d = [Int]()



```

2. Create an immutable array containing 5 city names.



```
var str = "Hello, playground"


let a:[String] = ["Kolkata", "Delhi", "Mumbai", "Chennai",
    "Puri"]
print(a)
a.append(newElement: String)
```

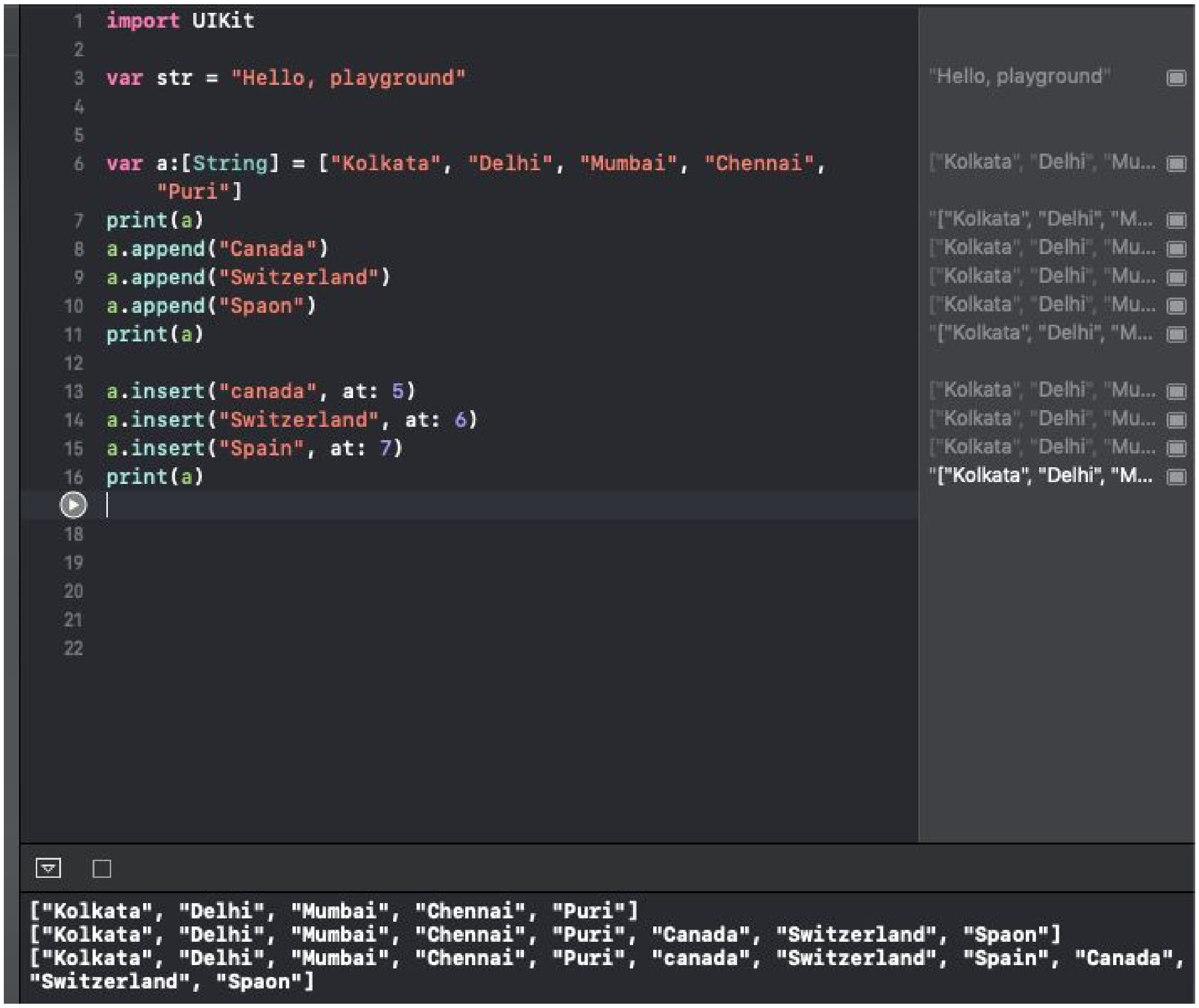
Cannot use mutating member on immutable value: 'a' is a 'let' constant

Change 'let' to 'var' to make it mutable

Editor placeholder in source file

```
["Kolkata", "Delhi", "Mumbai", "Chennai", "Puri"]
```

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.

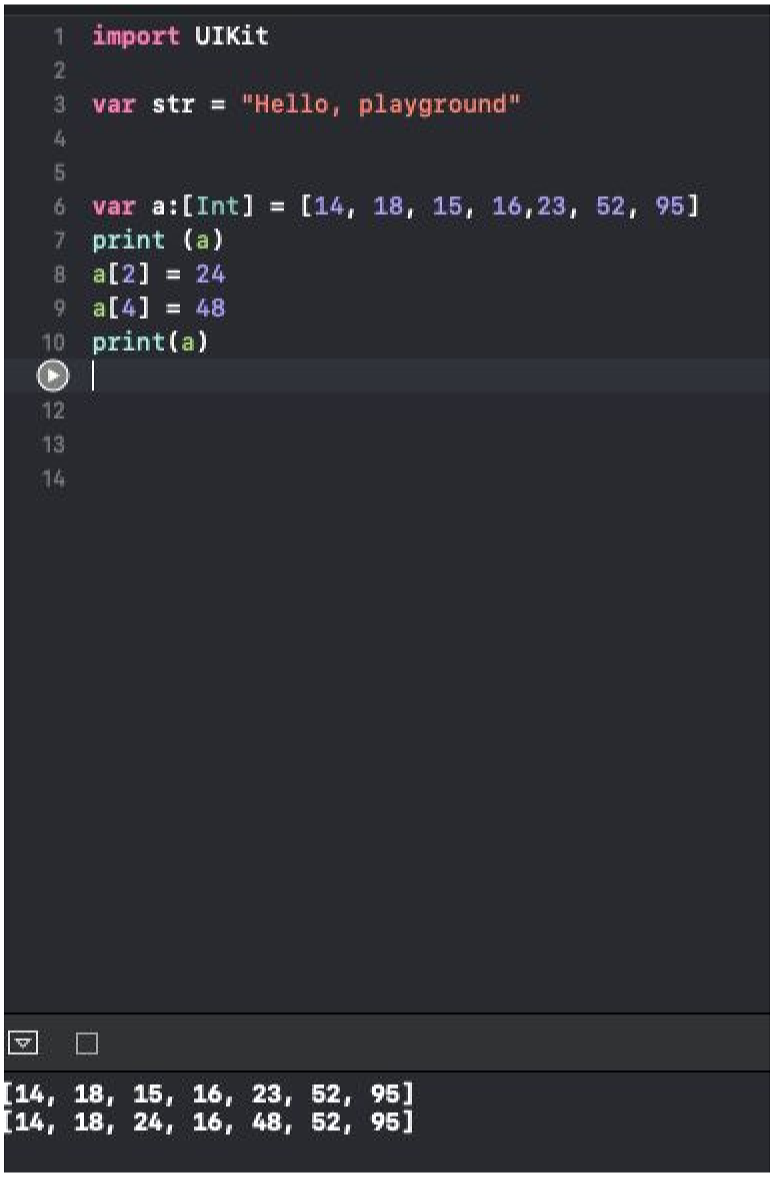


```swift
import UIKit

var str = "Hello, playground"


var a:[String] = ["Kolkata", "Delhi", "Mumbai", "Chennai",
    "Puri"]
print(a)
a.append("Canada")
a.append("Switzerland")
a.append("Spaon")
print(a)

a.insert("canada", at: 5)
a.insert("Switzerland", at: 6)
a.insert("Spain", at: 7)
print(a)
```

```
"Hello, playground"
["Kolkata", "Delhi", "Mu...
"["Kolkata", "Delhi", "M...
["Kolkata", "Delhi", "Mu...
["Kolkata", "Delhi", "Mu...
["Kolkata", "Delhi", "Mu...
"["Kolkata", "Delhi", "M...
["Kolkata", "Delhi", "Mu...
["Kolkata", "Delhi", "Mu...
["Kolkata", "Delhi", "Mu...
"["Kolkata", "Delhi", "M...
```

```
["Kolkata", "Delhi", "Mumbai", "Chennai", "Puri"]
["Kolkata", "Delhi", "Mumbai", "Chennai", "Puri", "Canada", "Switzerland", "Spaon"]
["Kolkata", "Delhi", "Mumbai", "Chennai", "Puri", "canada", "Switzerland", "Spain", "Canada",
"Switzerland", "Spaon"]
```

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



```swift
import UIKit

var str = "Hello, playground"


var a:[Int] = [14, 18, 15, 16,23, 52, 95]
print (a)
a[2] = 24
a[4] = 48
print(a)
```

```
[14, 18, 15, 16, 23, 52, 95]
[14, 18, 24, 16, 48, 52, 95]
```

//Set

1. Given the following sets:

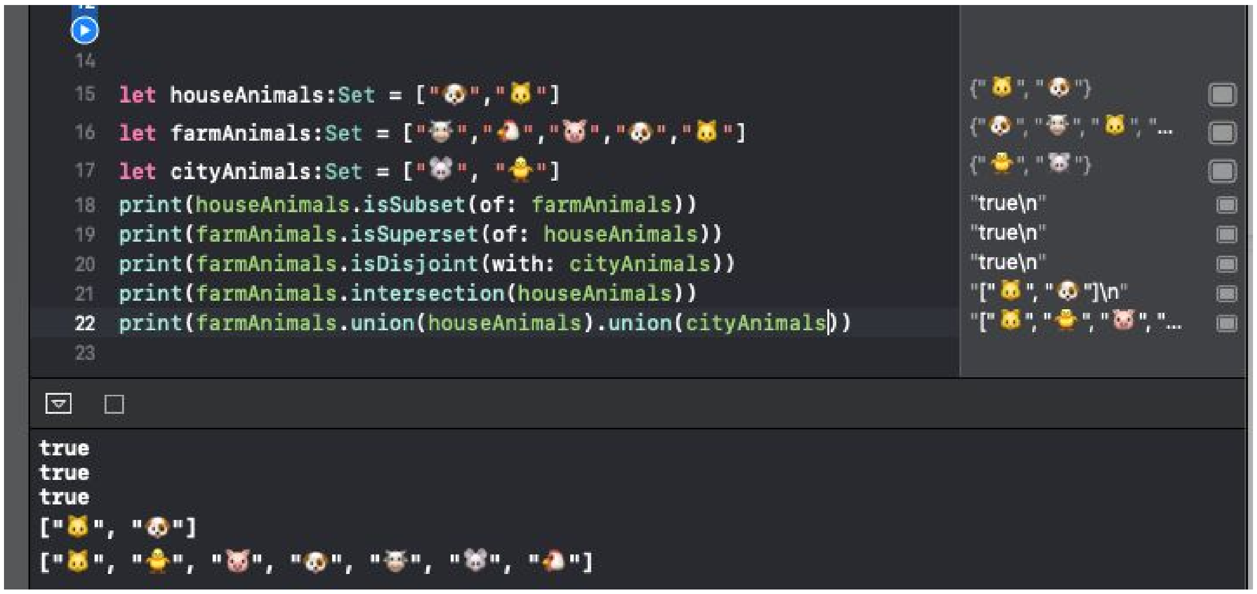
let houseAnimals: Set = ["🐶", "🐱"]

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

let cityAnimals: Set = ["🐦", "🐭"]

Use set operations to...

1. Determine whether the set of house animals is a subset of farm animals.
2. Determine whether the set of farm animals is a superset of house animals.
3. Determine if the set of farm animals is disjoint with city animals.
4. Create a set that only contains farm animals that are not also house animals.
5. Create a set that contains all the animals from all sets.



```swift
let houseAnimals:Set = ["🐶","🐱"]
let farmAnimals:Set = ["🐮","🐔","🐷","🐶","🐱"]
let cityAnimals:Set = ["🐭", "🐥"]
print(houseAnimals.isSubset(of: farmAnimals))
print(farmAnimals.isSuperset(of: houseAnimals))
print(farmAnimals.isDisjoint(with: cityAnimals))
print(farmAnimals.intersection(houseAnimals))
print(farmAnimals.union(houseAnimals).union(cityAnimals))
```

```
{"🐱", "🐶"}
{"🐶", "🐮", "🐱", "...
{"🐥", "🐭"}
"true\n"
"true\n"
"true\n"
"["🐱", "🐶"]\n"
"["🐱", "🐥", "🐷", "...
```

```
true
true
true
["🐱", "🐶"]
["🐱", "🐥", "🐷", "🐶", "🐮", "🐭", "🐔"]
```

// 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).

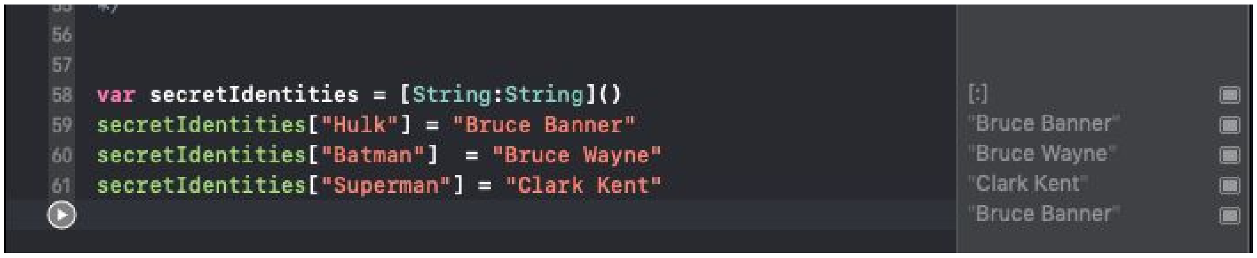
   var myDictionary1: Dictionary<String, Int> = [:]

   var myDictionary2: [String: Int] = [:]

   var myDictionary3 = Dictionary<String, Int>()

   var myDictionary4 = [String: Int]()

2. Create a mutable dictionary named secretIdentities where the key value pairs are "Hulk" -> "Bruce Banner", "Batman" -> "Bruce Wayne", and "Superman" -> "Clark Kent".

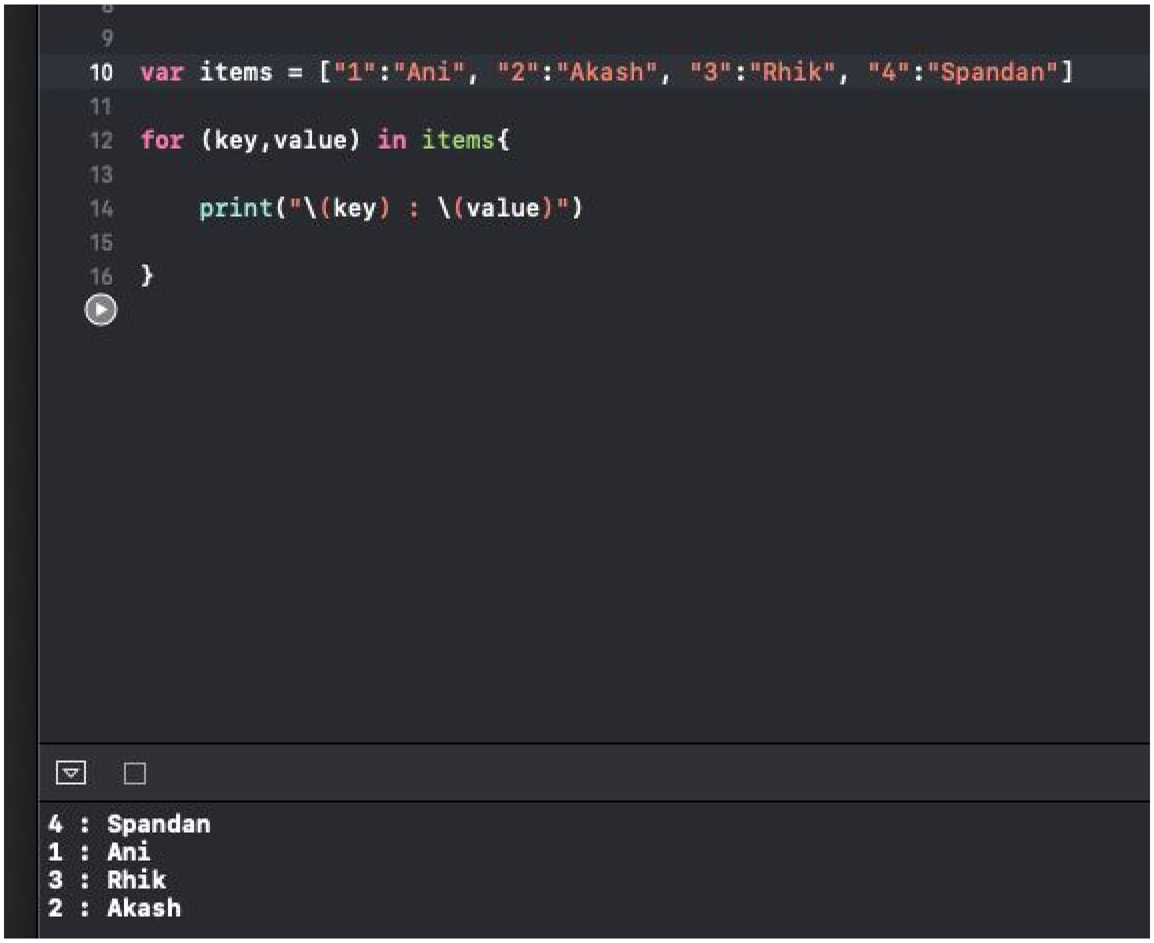


```
var secretIdentities = [String:String]()
secretIdentities["Hulk"] = "Bruce Banner"
secretIdentities["Batman"]  = "Bruce Wayne"
secretIdentities["Superman"] = "Clark Kent"
```

```
[:]
"Bruce Banner"
"Bruce Wayne"
"Clark Kent"
"Bruce Banner"
```

3. Create a nesters structure of Key-value pair.

4. Print all the keys in the dic



```
var items = ["1":"Ani", "2":"Akash", "3":"Rhik", "4":"Spandan"]

for (key,value) in items{

    print("\(key) : \(value)")

}
```

```
4 : Spandan
1 : Ani
3 : Rhik
2 : Akash
```

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. You use subscripts to set and retrieve values by index without needing separate methods for setting and retrieval. For example, you access elements in an Array instance as some Array[index] and elements in a Dictionary instance as someDictionary[key].

```
subscript(index: Int) -> Int {
get {
//code

}

set(newValue) {

//code

   }

}
```

2. Create a simple subscript that outputs true if a string contains a substring and false otherwise.