

**Exercise :**  
**// Initializers**

1. Implement the parameterised initialisation with class or struct.



```
UIKit
Related Items
3 var str = "Hello, playground"
4 class Person {
5     var name: String?
6     var sur: String?
7
8     init (name: String, sur:String) {
9         self.name = name
10        self.sur = sur
11    }
12 }
13 var p = Person(name: "Anindya", sur: "Guha")
14 print(p.name!)
15 print(p.sur!)
17
```

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

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

```
sample2002

1  import UIKit
2
3  var str = "Hello, playground"
4  class MOVIE {
5      var id: Int = 12345
6      var name: String
7      var title:String = "Nature"
8      var author: String
9      var publish_date:String
10
11     init (n: String, a:String, pub:String) {
12         self.name = n
13         self.author = a
14         self.publish_date = pub
15     }
16 }
17 convenience init() { ⚠ All paths through this function will call itself
18     self.init()
19 }
20 }
21 var p = MOVIE(n: "Hawaayein", a: "Guha", pub: "12/04/2018")
22 print(p.id)
23 print(p.name)
24 print(p.title)
25 print(p.author)
26 print(p.publish_date)
27
28
```

12345  
Hawaayein  
Nature  
Guha  
12/04/2018

4. Declare a structure which can demonstrate the throwable Initializer

```
Struct person {  
  
    Var name:String  
  
    init() throws  
  
    {  
  
        try{ //code}  
  
        catch{// code}  
  
    }  
}
```

## // Array

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

```
36     }  
37 }  
38 */  
39  
40 var a:Array<Int> = [1,2,3,4,5]  
41 var b:[Int] = [1,2,3,4,5]  
42 var c = [1,2,4,3,5]  
43 var d = [Int]()  
44  
45  
46
```

2. Create an immutable array containing 5 city names.

```
2
3 var str = "Hello, playground"
4
5
6 let a:[String] = ["Kolkata", "Delhi", "Mumbai", "Chennai",
7   "Puri"]
8 print(a)
9 a.append(newElement: String)
```

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

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

Fix

⚠ Editor placeholder in source file

```
14
15
16
```

☐ ☐

["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.

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



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

```
1  import UIKit
2
3  var str = "Hello, playground"
4
5
6  var a:[Int] = [14, 18, 15, 16,23, 52, 95]
7  print (a)
8  a[2] = 24
9  a[4] = 48
10 print(a)
11
12
13
14
```

[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.

```
14
15 let houseAnimals:Set = ["🐱", "🐱"]
16 let farmAnimals:Set = ["🐱", "🐱", "🐱", "🐱", "🐱", "🐱"]
17 let cityAnimals:Set = ["🐱", "🐱"]
18 print(houseAnimals.isSubset(of: farmAnimals))
19 print(farmAnimals.isSuperset(of: houseAnimals))
20 print(farmAnimals.isDisjoint(with: cityAnimals))
21 print(farmAnimals.intersection(houseAnimals))
22 print(farmAnimals.union(houseAnimals).union(cityAnimals))
23
```

Output:

```
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).

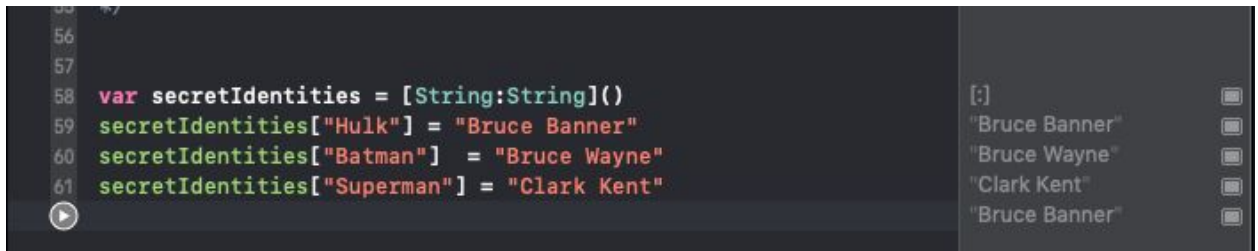
```
Var a = Dictionary<String,String>()
```

```
Var b = [int:int]()
```

```
Var c:[int:int] = [:]
```

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

```
56
57
58 var secretIdentities = [String:String]()
59 secretIdentities["Hulk"] = "Bruce Banner"
60 secretIdentities["Batman"] = "Bruce Wayne"
61 secretIdentities["Superman"] = "Clark Kent"
```

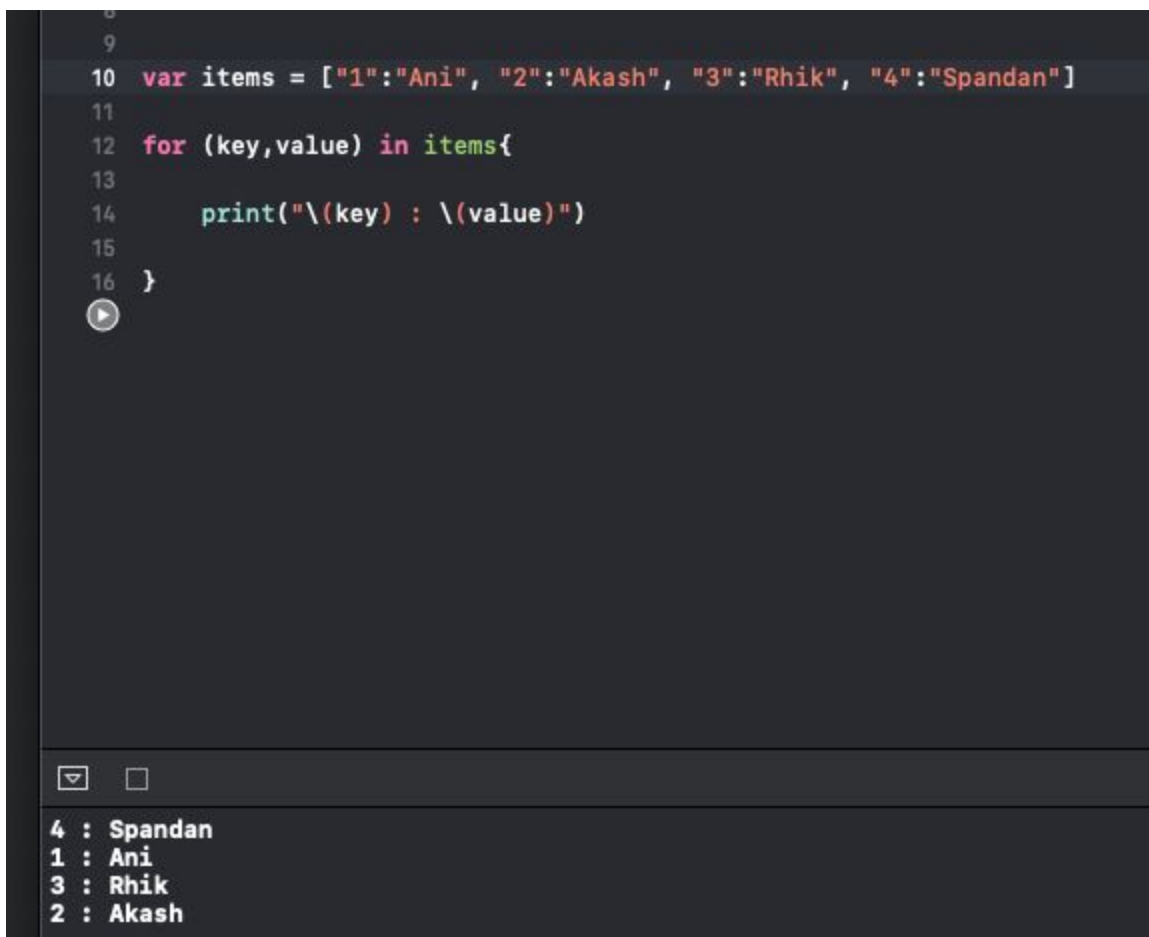
The screenshot shows a code editor with Swift code. The code defines a dictionary named 'secretIdentities' with three entries: 'Hulk' maps to 'Bruce Banner', 'Batman' maps to 'Bruce Wayne', and 'Superman' maps to 'Clark Kent'. To the right of the code, the dictionary's contents are displayed in a structured format, showing the keys and their corresponding values.

```
[:]
"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

```
8
9
10 var items = ["1":"Ani", "2":"Akash", "3":"Rhik", "4":"Spandan"]
11
12 for (key,value) in items{
13
14     print("\(key) : \(value)")
15
16 }
```

The screenshot shows a code editor with Swift code. The code defines an array named 'items' with four elements: "1":"Ani", "2":"Akash", "3":"Rhik", and "4":"Spandan". A for loop iterates over the array, printing each key-value pair. The console output at the bottom shows the results of the loop, with the keys and values printed in reverse order of the array.

```
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.