Date: 18.02.2021

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

SWIFT BASICS Assignment

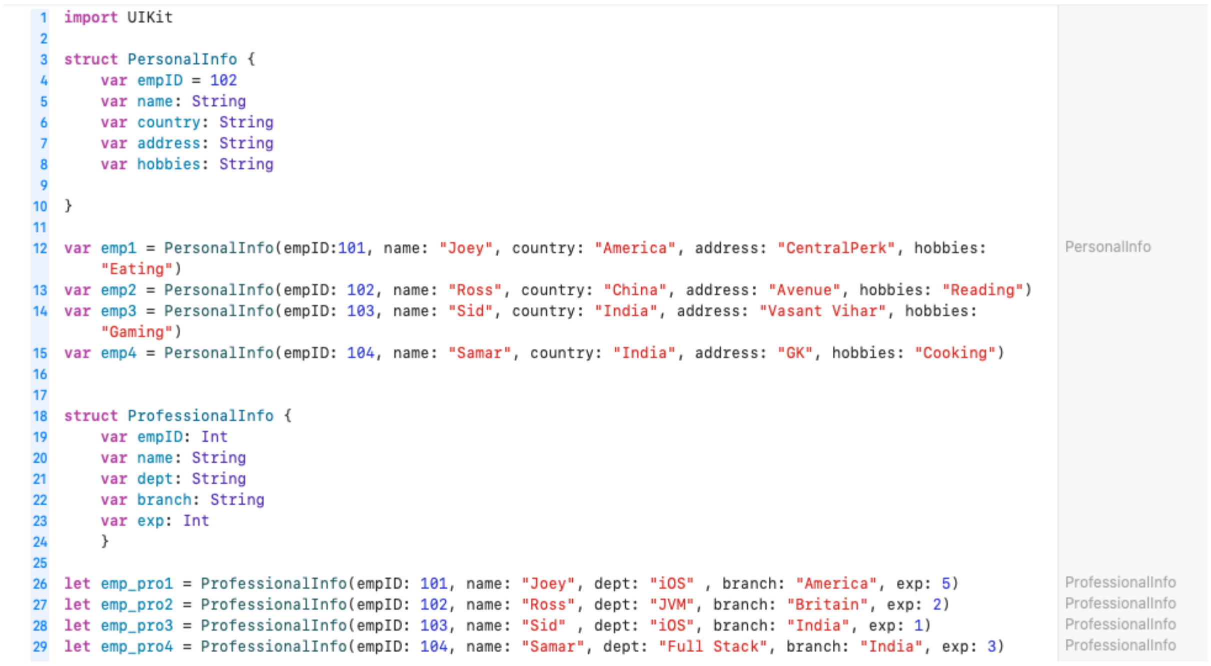
Exercise 1

1. Create a employee personal information structure and employee professional structure

the properties for personal :
employeeID
name
country(america,india,britain,japan,china)
address
hobbies(optional)

properties for professional

employeeID
name
department(iOS, android, jvm, full stack, web)
branch(america,india,britain,japan,china)
experience



```swift
import UIKit

struct PersonalInfo {
    var empID = 102
    var name: String
    var country: String
    var address: String
    var hobbies: String

}

var emp1 = PersonalInfo(empID:101, name: "Joey", country: "America", address: "CentralPerk", hobbies:
    "Eating")
var emp2 = PersonalInfo(empID: 102, name: "Ross", country: "China", address: "Avenue", hobbies: "Reading")
var emp3 = PersonalInfo(empID: 103, name: "Sid", country: "India", address: "Vasant Vihar", hobbies:
    "Gaming")
var emp4 = PersonalInfo(empID: 104, name: "Samar", country: "India", address: "GK", hobbies: "Cooking")


struct ProfessionalInfo {
    var empID: Int
    var name: String
    var dept: String
    var branch: String
    var exp: Int
    }

let emp_pro1 = ProfessionalInfo(empID: 101, name: "Joey", dept: "iOS" , branch: "America", exp: 5)
let emp_pro2 = ProfessionalInfo(empID: 102, name: "Ross", dept: "JVM", branch: "Britain", exp: 2)
let emp_pro3 = ProfessionalInfo(empID: 103, name: "Sid" , dept: "iOS", branch: "India", exp: 1)
let emp_pro4 = ProfessionalInfo(empID: 104, name: "Samar", dept: "Full Stack", branch: "India", exp: 3)
```

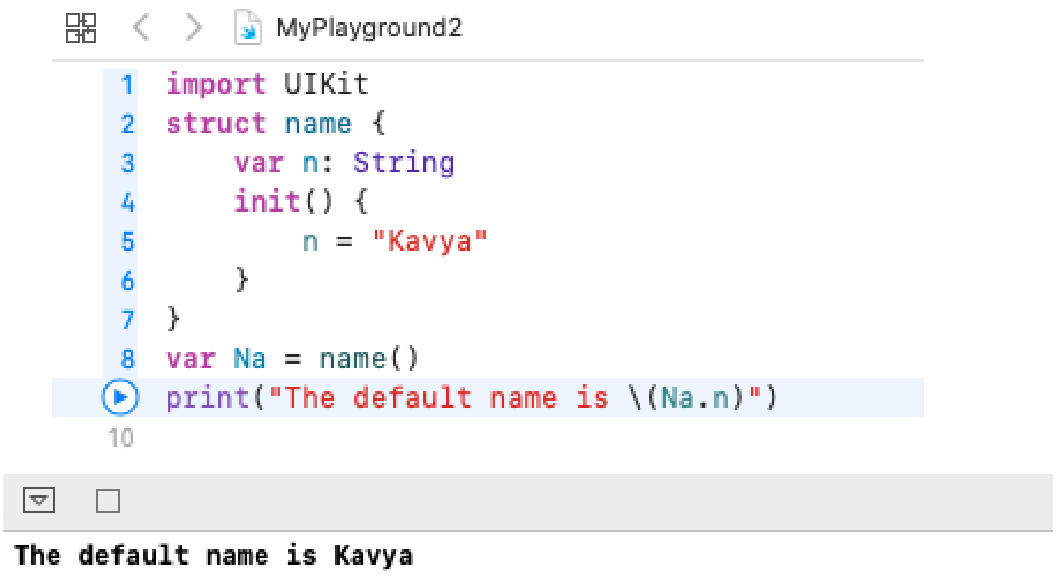
```
PersonalInfo
ProfessionalInfo
ProfessionalInfo
ProfessionalInfo
ProfessionalInfo
```

(Confusion in creating the functions inside struct for the rest of the parts)

<u>Exercise 2</u>

<u>INITIALIZERS</u>

1. Implement the parameterised initialisation with class or struct.



```
import UIKit
struct name {
    var n: String
    init() {
        n = "Kavya"
    }
}
var Na = name()
print("The default name is \(Na.n)")
```

```
The default name is Kavya
```

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.

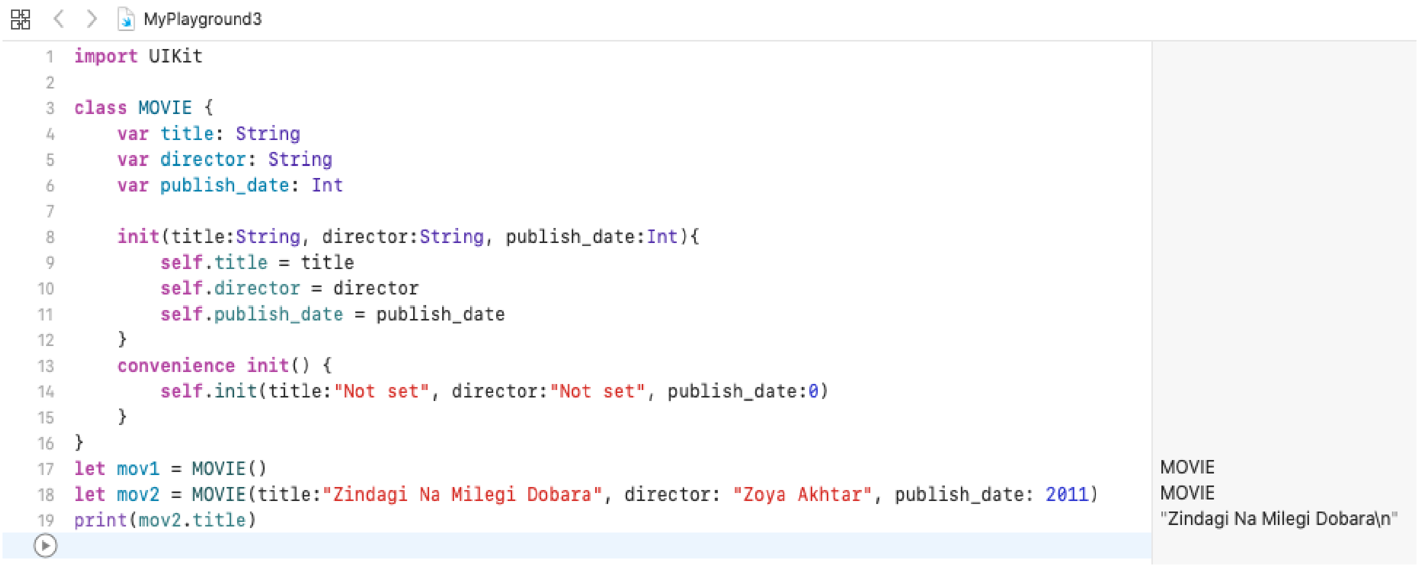
Rule 4: A designated initializer must ensure that all of the properties introduced by its class are initialized before it delegates up to a superclass initializer.

Rule 5: A designated initializer must delegate up to a superclass initializer before assigning a value to an inherited property. If it doesn't, the new value the designated initializer assigns will be overwritten by the superclass as part of its own initialization.

Rule 6: A convenience initializer must delegate to another initializer before assigning a value to *any* property (including properties defined by the same class). If it doesn't, the new value the convenience initializer assigns will be overwritten by its own class's designated initializer.

Rule 7: An initializer cannot call any instance methods, read the values of any instance properties, or refer to self as a value until after the first phase of initialization is complete.

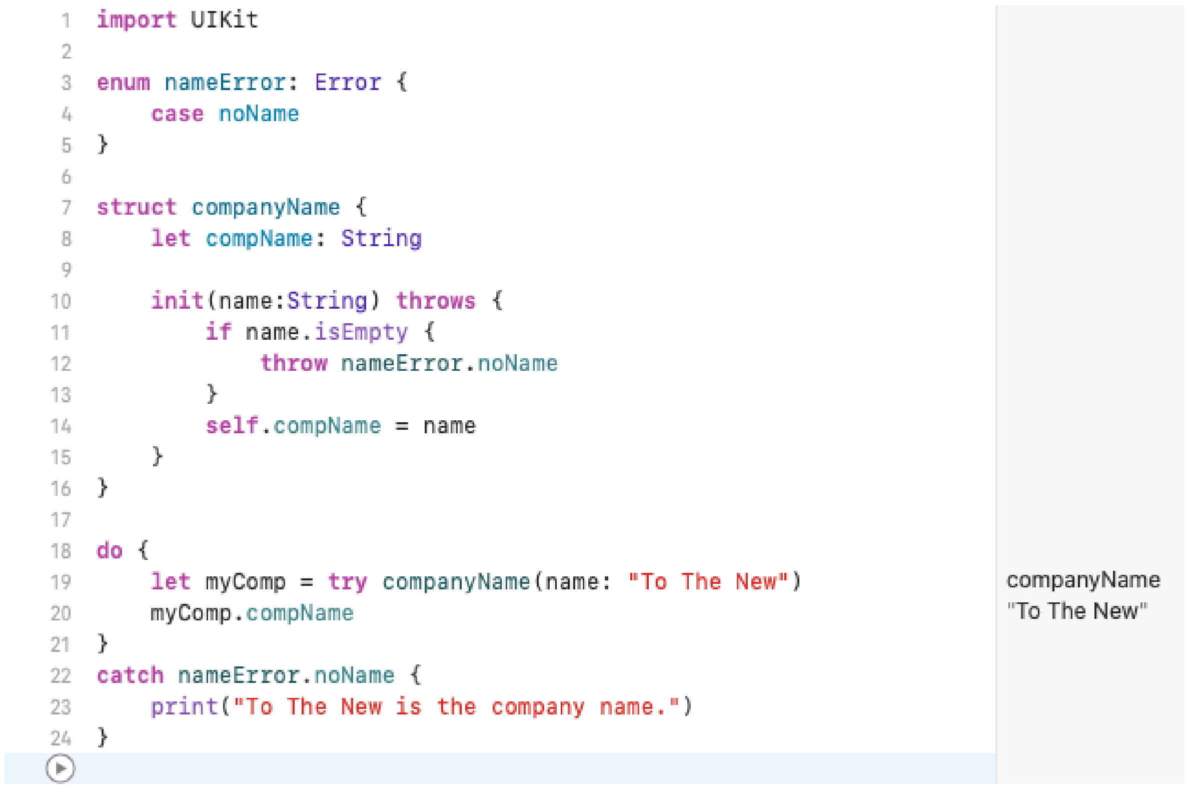
3. Using convenience **Initializers**, write-down the **Initializers** for MOVIE class having basic attributes like title, director, publish_date, etc.



```swift
import UIKit

class MOVIE {
    var title: String
    var director: String
    var publish_date: Int

    init(title:String, director:String, publish_date:Int){
        self.title = title
        self.director = director
        self.publish_date = publish_date
    }
    convenience init() {
        self.init(title:"Not set", director:"Not set", publish_date:0)
    }
}
let mov1 = MOVIE()
let mov2 = MOVIE(title:"Zindagi Na Milegi Dobara", director: "Zoya Akhtar", publish_date: 2011)
print(mov2.title)
```

```
MOVIE
MOVIE
"Zindagi Na Milegi Dobara\n"
```

4. Declare a structure which can demonstrate the throwable Initializer.



```swift
import UIKit

enum nameError: Error {
    case noName
}

struct companyName {
    let compName: String

    init(name:String) throws {
        if name.isEmpty {
            throw nameError.noName
        }
        self.compName = name
    }
}

do {
    let myComp = try companyName(name: "To The New")
    myComp.compName
}
catch nameError.noName {
    print("To The New is the company name.")
}
```

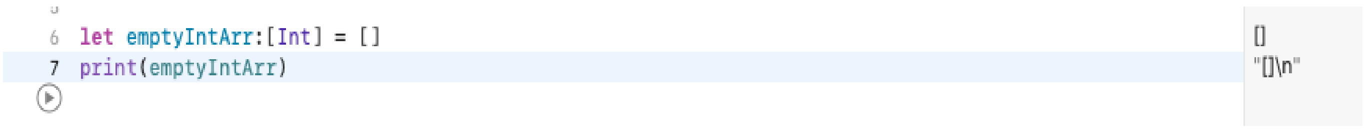
```
companyName
"To The New"
```

ARRAYS

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

Way 1

Declaring an empty array

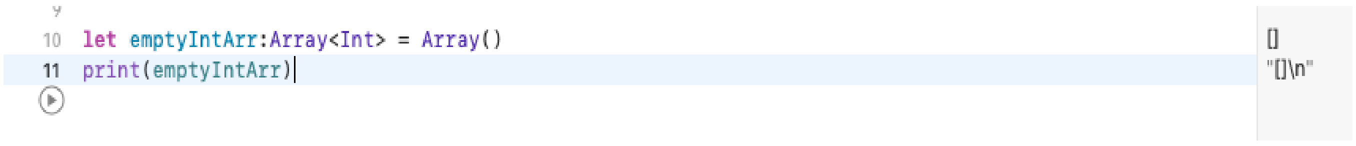


```swift
let emptyIntArr:[Int] = []
print(emptyIntArr)
```

```
[]
"[]\n"
```

In the above program, we have declared a constant emptyIntArr that can store array of integer and initialized with 0 values.

Way 2

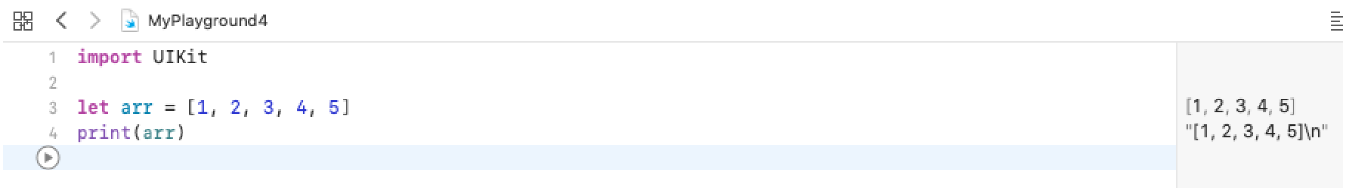


```swift
let emptyIntArr:Array<Int> = Array()
print(emptyIntArr)
```

```
[]
"[]\n"
```

Way 3

Declaring an array with some values

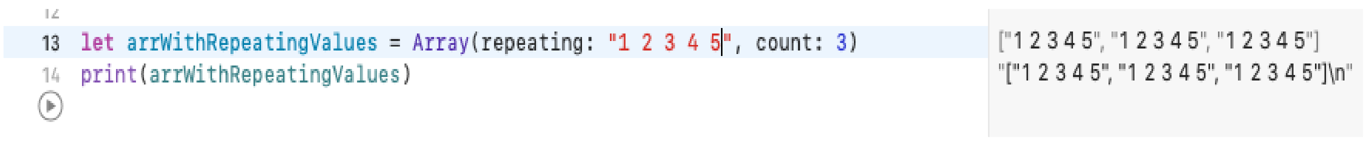

MyPlayground4

```
import UIKit

let arr = [1, 2, 3, 4, 5]
print(arr)
```

```
[1, 2, 3, 4, 5]
"[1, 2, 3, 4, 5]\n"
```

Way 4

Declaring an array containing the specified number of a single repeated value

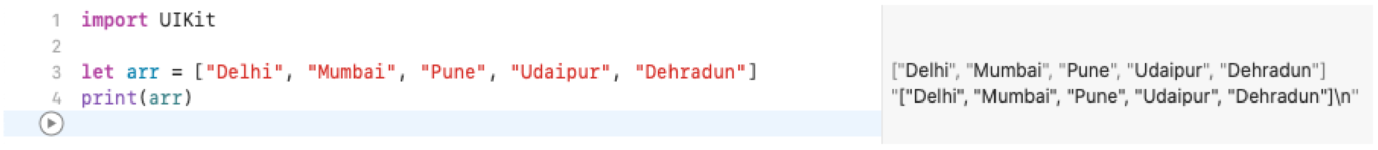


```
let arrWithRepeatingValues = Array(repeating: "1 2 3 4 5", count: 3)
print(arrWithRepeatingValues)
```

```
["1 2 3 4 5", "1 2 3 4 5", "1 2 3 4 5"]
"["1 2 3 4 5", "1 2 3 4 5", "1 2 3 4 5"]\n"
```

2. Create an immutable array containing 5 city names.

Swift arrays are immutable if we define them as constants with let.

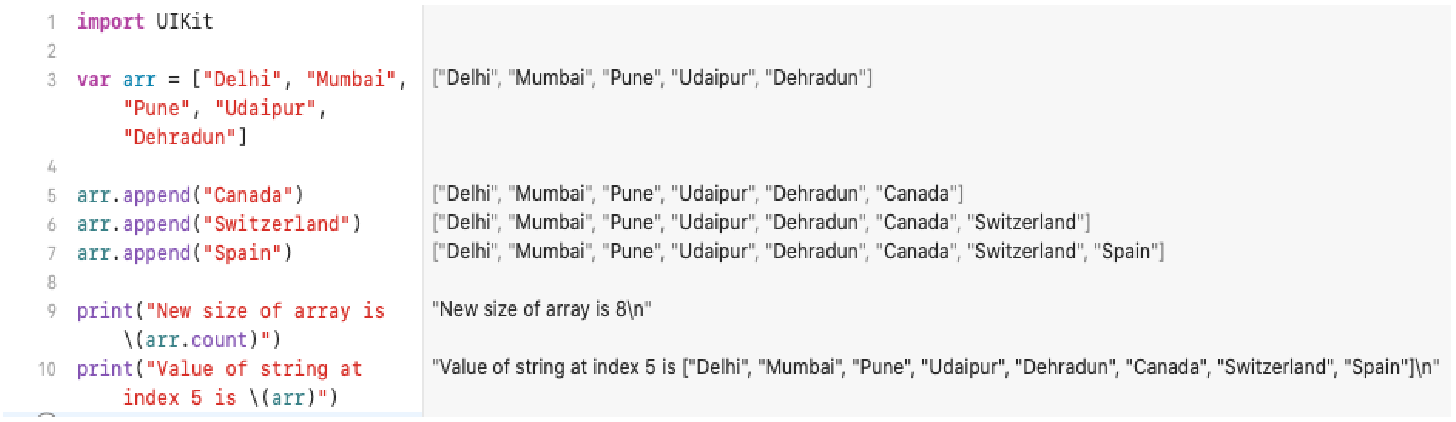


```
import UIKit

let arr = ["Delhi", "Mumbai", "Pune", "Udaipur", "Dehradun"]
print(arr)
```

```
["Delhi", "Mumbai", "Pune", "Udaipur", "Dehradun"]
"["Delhi", "Mumbai", "Pune", "Udaipur", "Dehradun"]\n"
```

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.

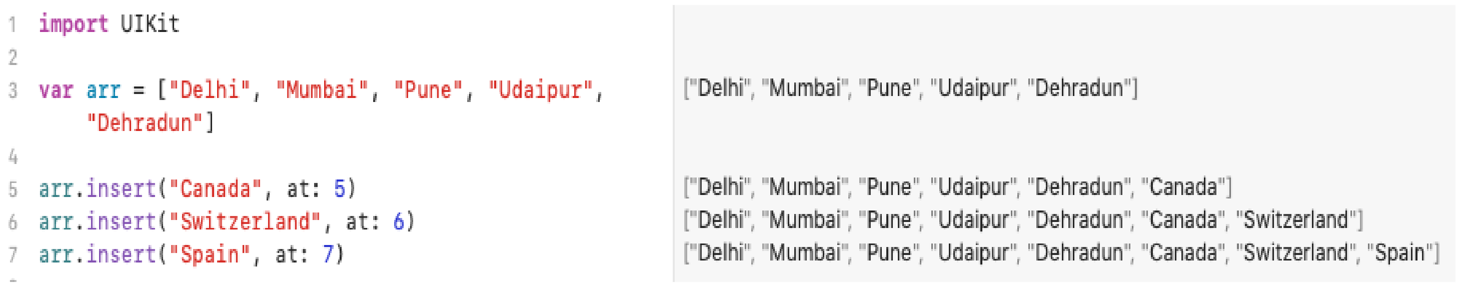
Method 1:



```swift
import UIKit

var arr = ["Delhi", "Mumbai",
    "Pune", "Udaipur",
    "Dehradun"]

arr.append("Canada")
arr.append("Switzerland")
arr.append("Spain")

print("New size of array is
    \(arr.count)")
print("Value of string at
    index 5 is \(arr)")
```

```
["Delhi", "Mumbai", "Pune", "Udaipur", "Dehradun"]

["Delhi", "Mumbai", "Pune", "Udaipur", "Dehradun", "Canada"]
["Delhi", "Mumbai", "Pune", "Udaipur", "Dehradun", "Canada", "Switzerland"]
["Delhi", "Mumbai", "Pune", "Udaipur", "Dehradun", "Canada", "Switzerland", "Spain"]

"New size of array is 8\n"

"Value of string at index 5 is ["Delhi", "Mumbai", "Pune", "Udaipur", "Dehradun", "Canada", "Switzerland", "Spain"]\n"
```

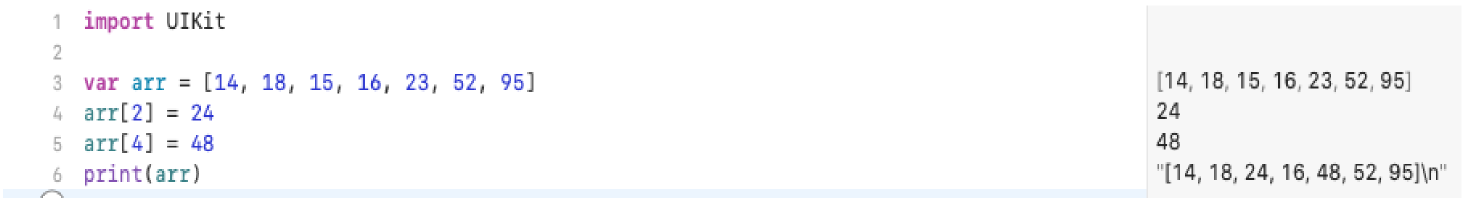
Method 2:



```swift
import UIKit

var arr = ["Delhi", "Mumbai", "Pune", "Udaipur",
    "Dehradun"]

arr.insert("Canada", at: 5)
arr.insert("Switzerland", at: 6)
arr.insert("Spain", at: 7)
```

```
["Delhi", "Mumbai", "Pune", "Udaipur", "Dehradun"]

["Delhi", "Mumbai", "Pune", "Udaipur", "Dehradun", "Canada"]
["Delhi", "Mumbai", "Pune", "Udaipur", "Dehradun", "Canada", "Switzerland"]
["Delhi", "Mumbai", "Pune", "Udaipur", "Dehradun", "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.



```swift
import UIKit

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

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

<u>SETS</u>

1. Given the following sets:

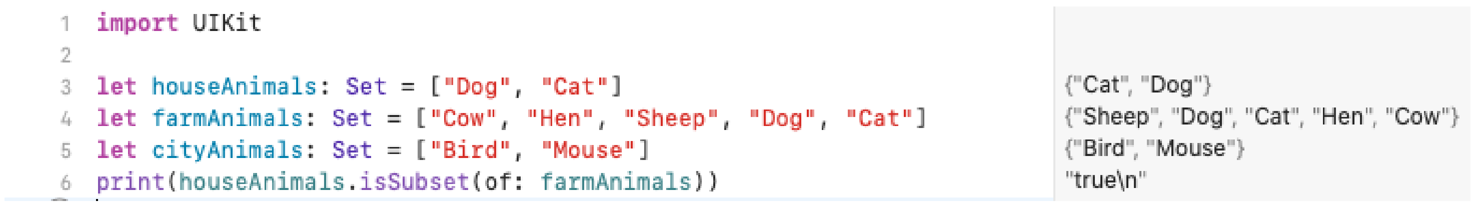
let houseAnimals: Set = ["🐶", "🐱"]

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

let cityAnimals: Set = ["🐦", "🐭"]

Use set operations to...

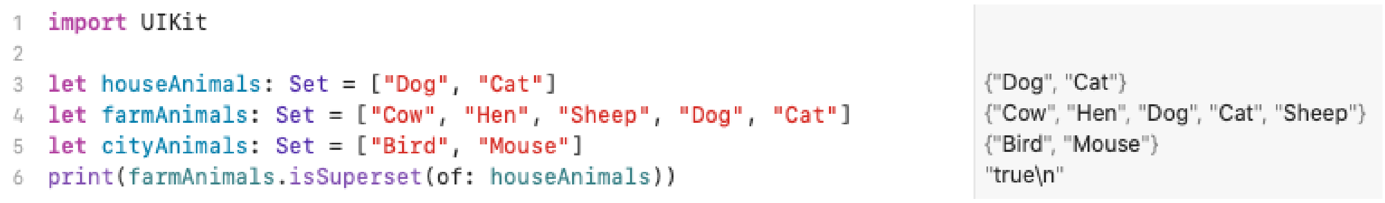
1. Determine whether the set of house animals is a subset of farm animals.



```
import UIKit

let houseAnimals: Set = ["Dog", "Cat"]
let farmAnimals: Set = ["Cow", "Hen", "Sheep", "Dog", "Cat"]
let cityAnimals: Set = ["Bird", "Mouse"]
print(houseAnimals.isSubset(of: farmAnimals))
```

```
{"Cat", "Dog"}
{"Sheep", "Dog", "Cat", "Hen", "Cow"}
{"Bird", "Mouse"}
"true\n"
```

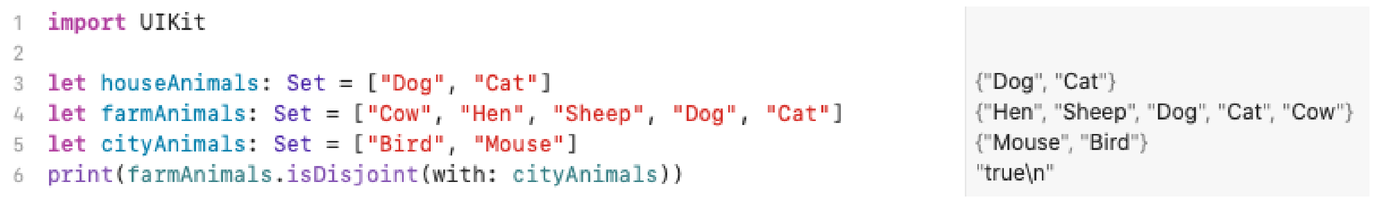
2. Determine whether the set of farm animals is a superset of house animals.



```
import UIKit

let houseAnimals: Set = ["Dog", "Cat"]
let farmAnimals: Set = ["Cow", "Hen", "Sheep", "Dog", "Cat"]
let cityAnimals: Set = ["Bird", "Mouse"]
print(farmAnimals.isSuperset(of: houseAnimals))
```

```
{"Dog", "Cat"}
{"Cow", "Hen", "Dog", "Cat", "Sheep"}
{"Bird", "Mouse"}
"true\n"
```

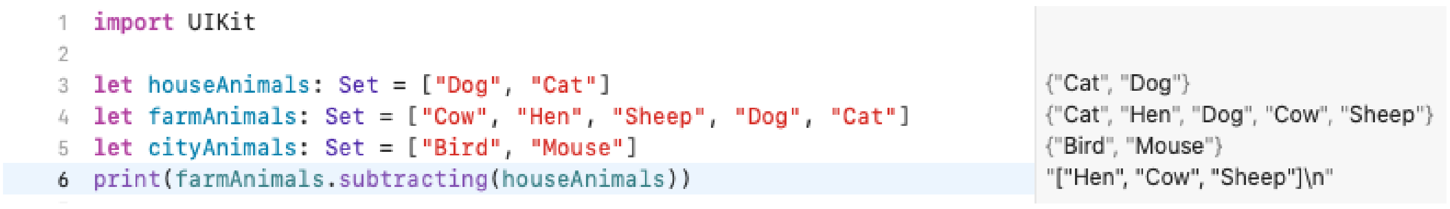
3. Determine if the set of farm animals is disjoint with city animals.



```
import UIKit

let houseAnimals: Set = ["Dog", "Cat"]
let farmAnimals: Set = ["Cow", "Hen", "Sheep", "Dog", "Cat"]
let cityAnimals: Set = ["Bird", "Mouse"]
print(farmAnimals.isDisjoint(with: cityAnimals))
```

```
{"Dog", "Cat"}
{"Hen", "Sheep", "Dog", "Cat", "Cow"}
{"Mouse", "Bird"}
"true\n"
```

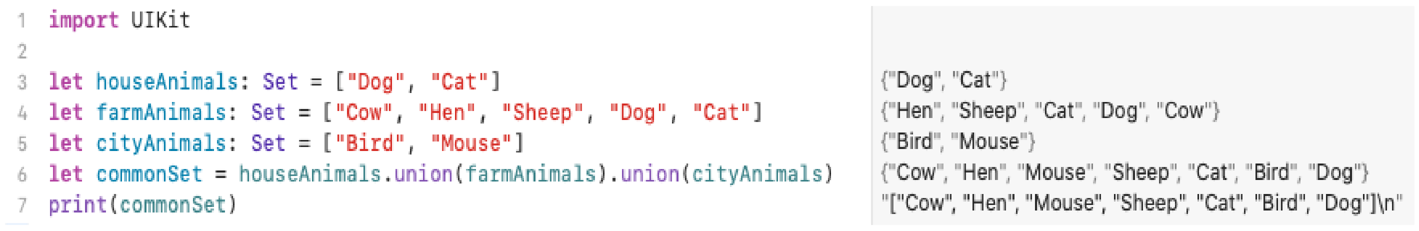
4. Create a set that only contains farm animals that are not also house animals.



```swift
import UIKit

let houseAnimals: Set = ["Dog", "Cat"]
let farmAnimals: Set = ["Cow", "Hen", "Sheep", "Dog", "Cat"]
let cityAnimals: Set = ["Bird", "Mouse"]
print(farmAnimals.subtracting(houseAnimals))
```

```
{"Cat", "Dog"}
{"Cat", "Hen", "Dog", "Cow", "Sheep"}
{"Bird", "Mouse"}
"["Hen", "Cow", "Sheep"]\n"
```

5. Create a set that contains all the animals from all sets.



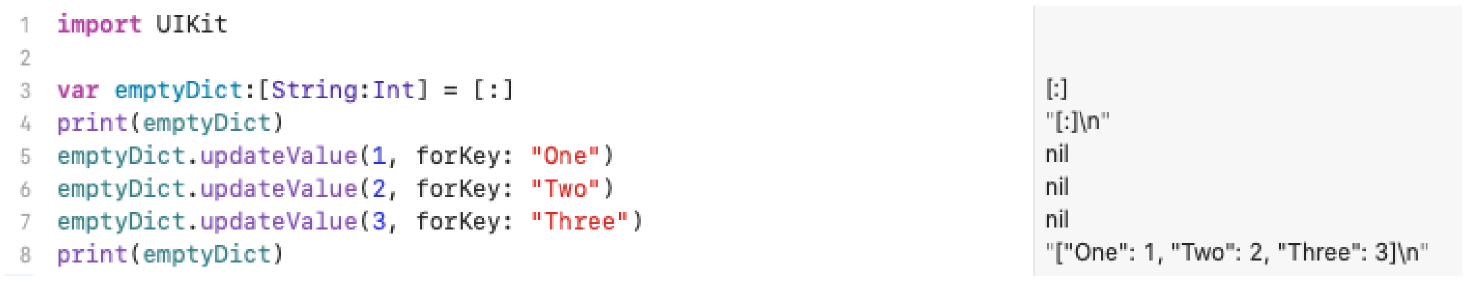
```swift
import UIKit

let houseAnimals: Set = ["Dog", "Cat"]
let farmAnimals: Set = ["Cow", "Hen", "Sheep", "Dog", "Cat"]
let cityAnimals: Set = ["Bird", "Mouse"]
let commonSet = houseAnimals.union(farmAnimals).union(cityAnimals)
print(commonSet)
```

```
{"Dog", "Cat"}
{"Hen", "Sheep", "Cat", "Dog", "Cow"}
{"Bird", "Mouse"}
{"Cow", "Hen", "Mouse", "Sheep", "Cat", "Bird", "Dog"}
"["Cow", "Hen", "Mouse", "Sheep", "Cat", "Bird", "Dog"]\n"
```

<u>DICTIONARY</u>

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).

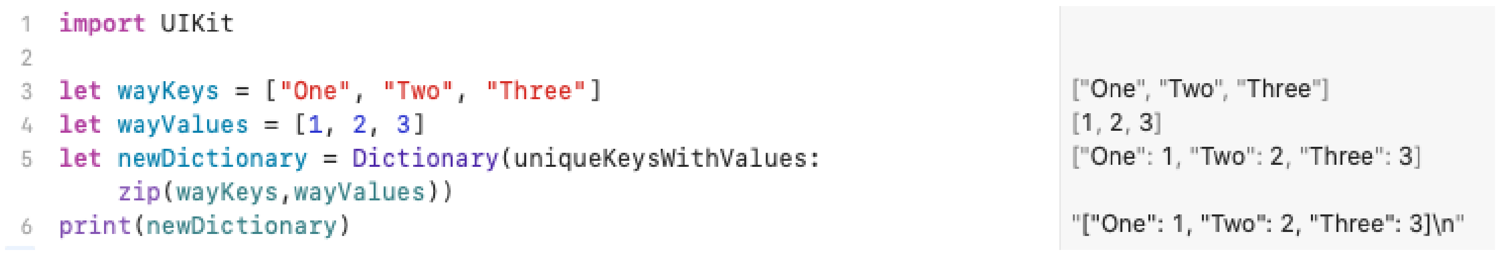
Way 1: Creating an empty Dictionary



```swift
import UIKit

var emptyDict:[String:Int] = [:]
print(emptyDict)
emptyDict.updateValue(1, forKey: "One")
emptyDict.updateValue(2, forKey: "Two")
emptyDict.updateValue(3, forKey: "Three")
print(emptyDict)
```

```
[:]
"[:]\n"
nil
nil
nil
"["One": 1, "Two": 2, "Three": 3]\n"
```

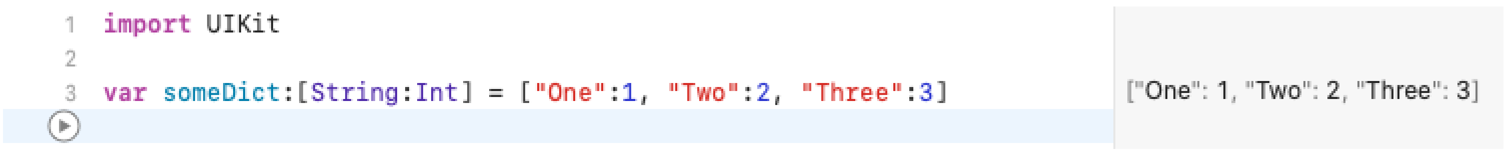
Way 2: Creating dictionary from two arrays



```swift
import UIKit

let wayKeys = ["One", "Two", "Three"]
let wayValues = [1, 2, 3]
let newDictionary = Dictionary(uniqueKeysWithValues:
    zip(wayKeys,wayValues))
print(newDictionary)
```

```
["One", "Two", "Three"]
[1, 2, 3]
["One": 1, "Two": 2, "Three": 3]

"["One": 1, "Two": 2, "Three": 3]\n"
```

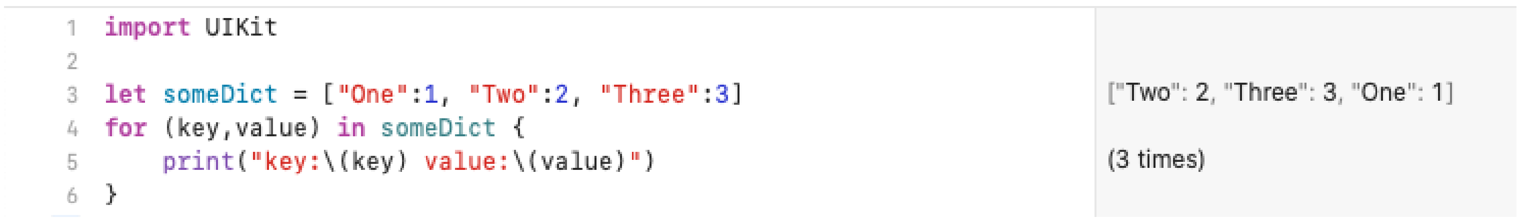
Way 3: Declaring an dictionary with some values



```swift
import UIKit

var someDict:[String:Int] = ["One":1, "Two":2, "Three":3]
```

```
["One": 1, "Two": 2, "Three": 3]
```

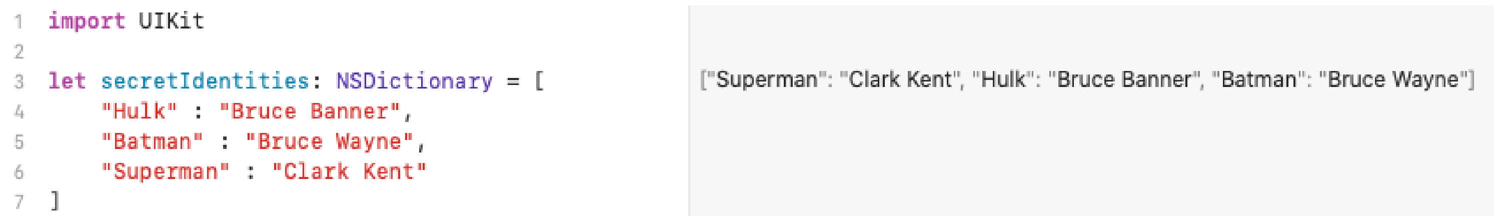
Way 4: Accessing elements of an dictionary with for-in loop



```swift
import UIKit

let someDict = ["One":1, "Two":2, "Three":3]
for (key,value) in someDict {
    print("key:\(key) value:\(value)")
}
```

```
["Two": 2, "Three": 3, "One": 1]

(3 times)
```

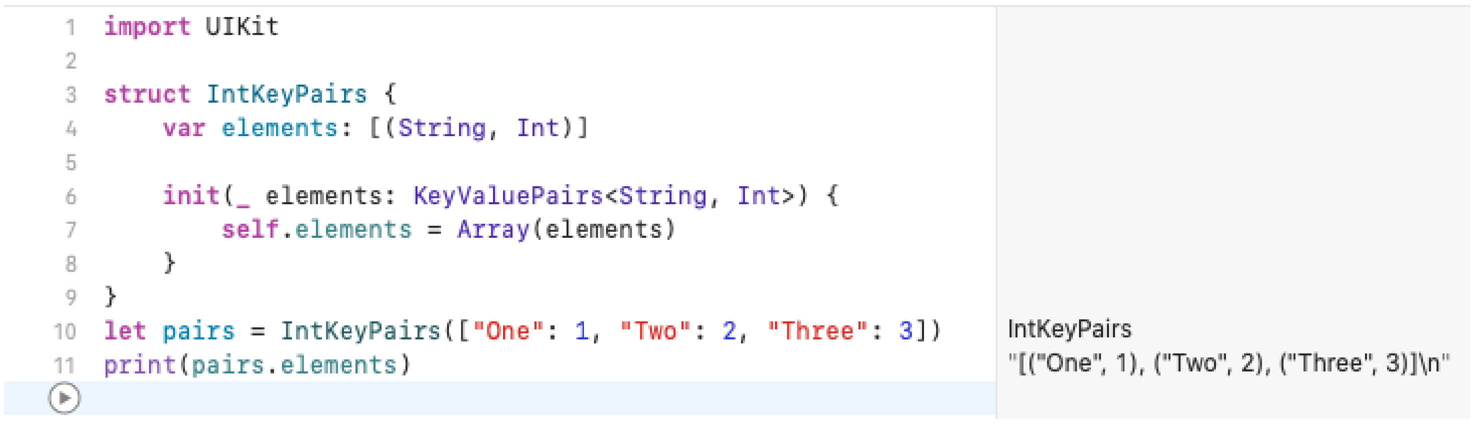
2. Create a mutable dictionary named secretIdentities where the key value pairs are "Hulk" -> "Bruce Banner", "Batman" -> "Bruce Wayne", and "Superman" -> "Clark Kent".



```swift
import UIKit

let secretIdentities: NSDictionary = [
    "Hulk" : "Bruce Banner",
    "Batman" : "Bruce Wayne",
    "Superman" : "Clark Kent"
]
```

```
["Superman": "Clark Kent", "Hulk": "Bruce Banner", "Batman": "Bruce Wayne"]
```

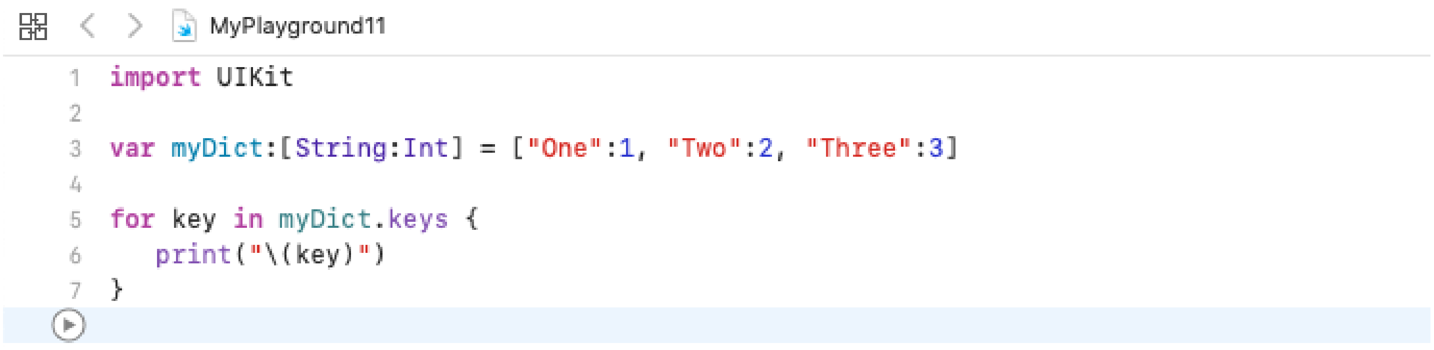
3. Create a nesters structure of Key-value pair.



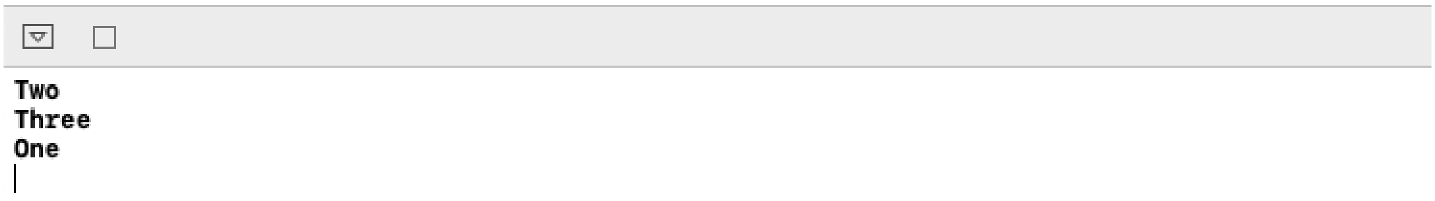
```swift
import UIKit

struct IntKeyPairs {
    var elements: [(String, Int)]

    init(_ elements: KeyValuePairs<String, Int>) {
        self.elements = Array(elements)
    }
}
let pairs = IntKeyPairs(["One": 1, "Two": 2, "Three": 3])
print(pairs.elements)
```

```
IntKeyPairs
"[("One", 1), ("Two", 2), ("Three", 3)]\n"
```

4. Print all the keys in the dictionary.



```
import UIKit

var myDict:[String:Int] = ["One":1, "Two":2, "Three":3]

for key in myDict.keys {
    print("\(key)")
}
```



```
Two
Three
One
```

<u>SUBSCRIPT</u>

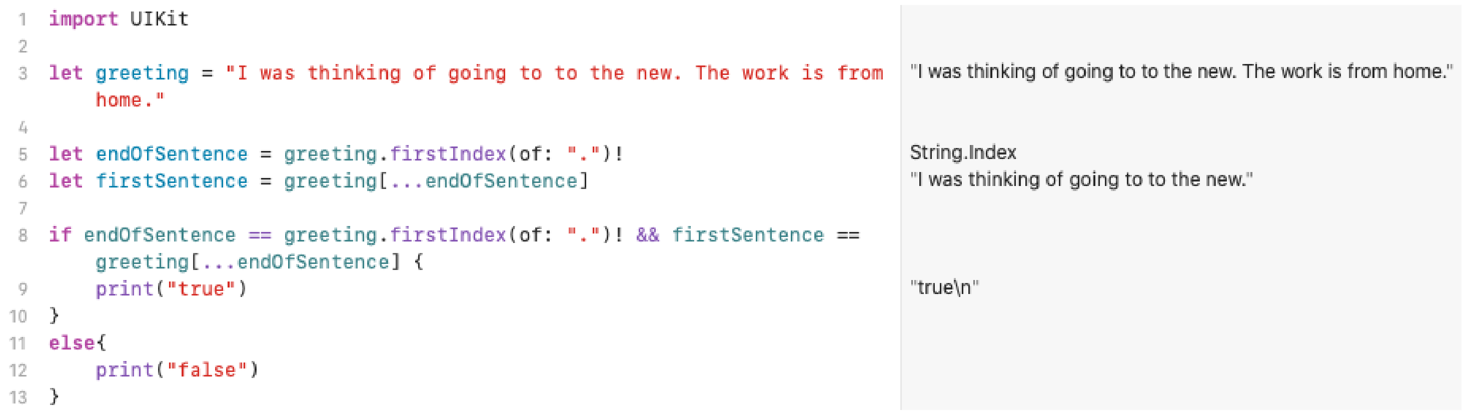
1. What is subscript ? Write down the declaration syntax.

A substring is a slice of a string. When you create a slice of a string, a Substring instance is the result. Operating on substrings is fast and efficient because a substring shares its storage with the original string. The Substring type presents the same interface as String, so you can avoid or defer any copying of the string’s contents.

Syntax:

```
subscript(index: Int) −> Int {
 get {
// used for subscript value declarations
 }
set(newValue) {
 // definitions are written here
 }
}
```

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



```swift
import UIKit

let greeting = "I was thinking of going to to the new. The work is from
    home."

let endOfSentence = greeting.firstIndex(of: ".")!
let firstSentence = greeting[...endOfSentence]

if endOfSentence == greeting.firstIndex(of: ".")! && firstSentence ==
    greeting[...endOfSentence] {
    print("true")
}
else{
    print("false")
}
```

```
"I was thinking of going to to the new. The work is from home."

String.Index
"I was thinking of going to to the new."

"true\n"
```