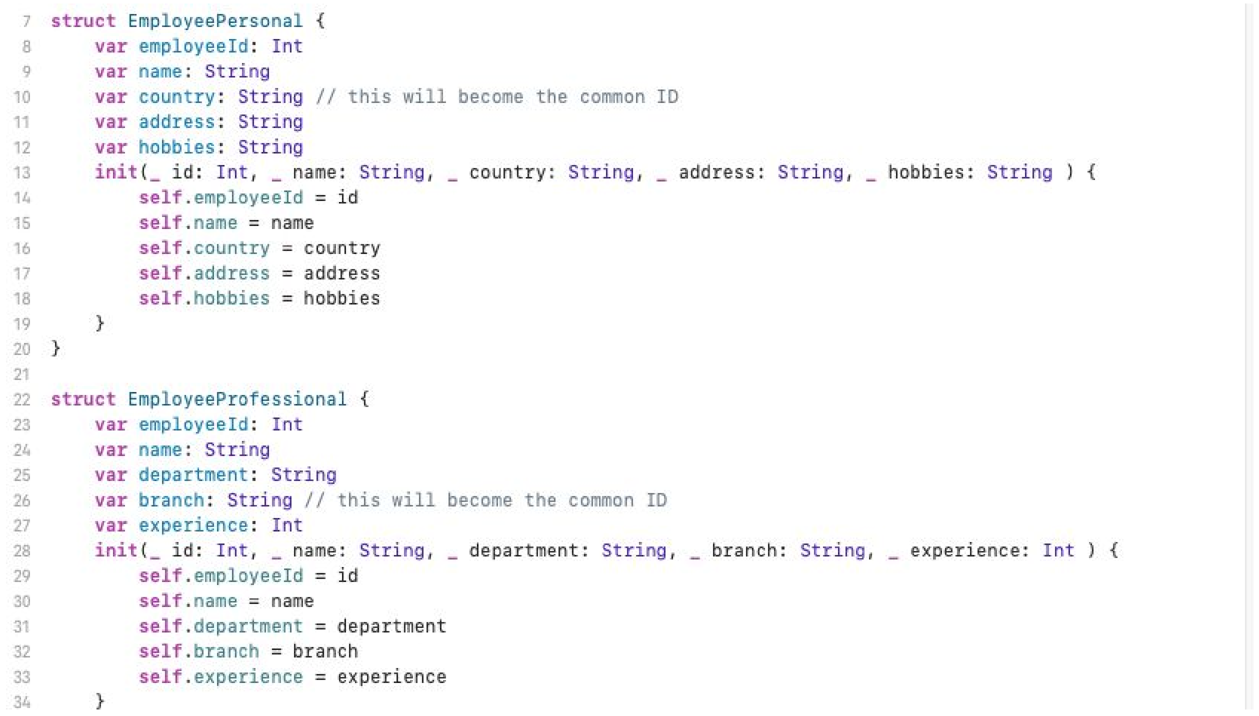
Exercise 1

Create a employee personal information structure and employee professional structure

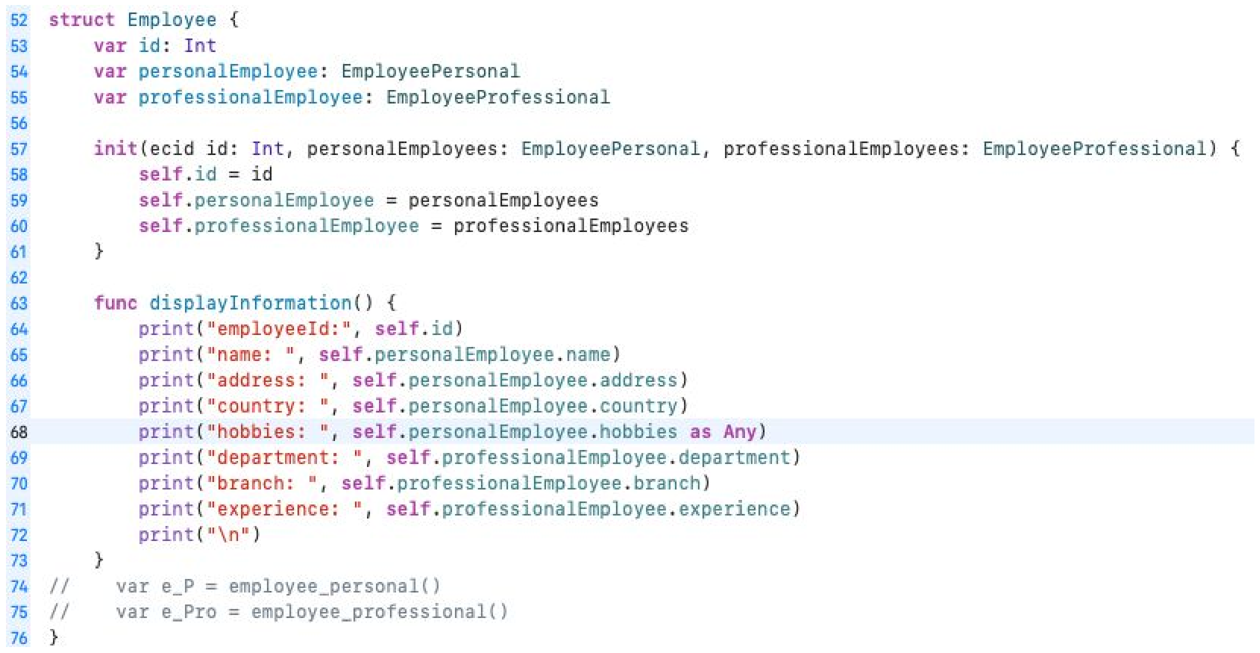
the properties for personal : employeeID, name, country(america,india,britain,japan,china), address, hobbies(optional),

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

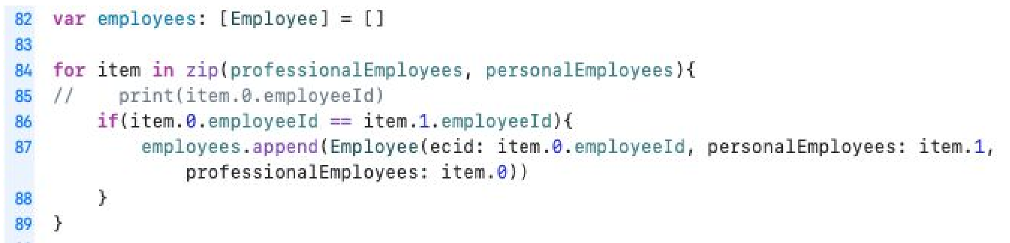


```swift
struct EmployeePersonal {
    var employeeId: Int
    var name: String
    var country: String // this will become the common ID
    var address: String
    var hobbies: String
    init(_ id: Int, _ name: String, _ country: String, _ address: String, _ hobbies: String ) {
        self.employeeId = id
        self.name = name
        self.country = country
        self.address = address
        self.hobbies = hobbies
    }
}

struct EmployeeProfessional {
    var employeeId: Int
    var name: String
    var department: String
    var branch: String // this will become the common ID
    var experience: Int
    init(_ id: Int, _ name: String, _ department: String, _ branch: String, _ experience: Int ) {
        self.employeeId = id
        self.name = name
        self.department = department
        self.branch = branch
        self.experience = experience
    }
```

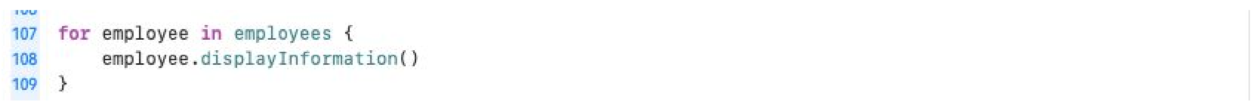
TASKS: 1. create a third employee structure that contains the information from both based on common id.



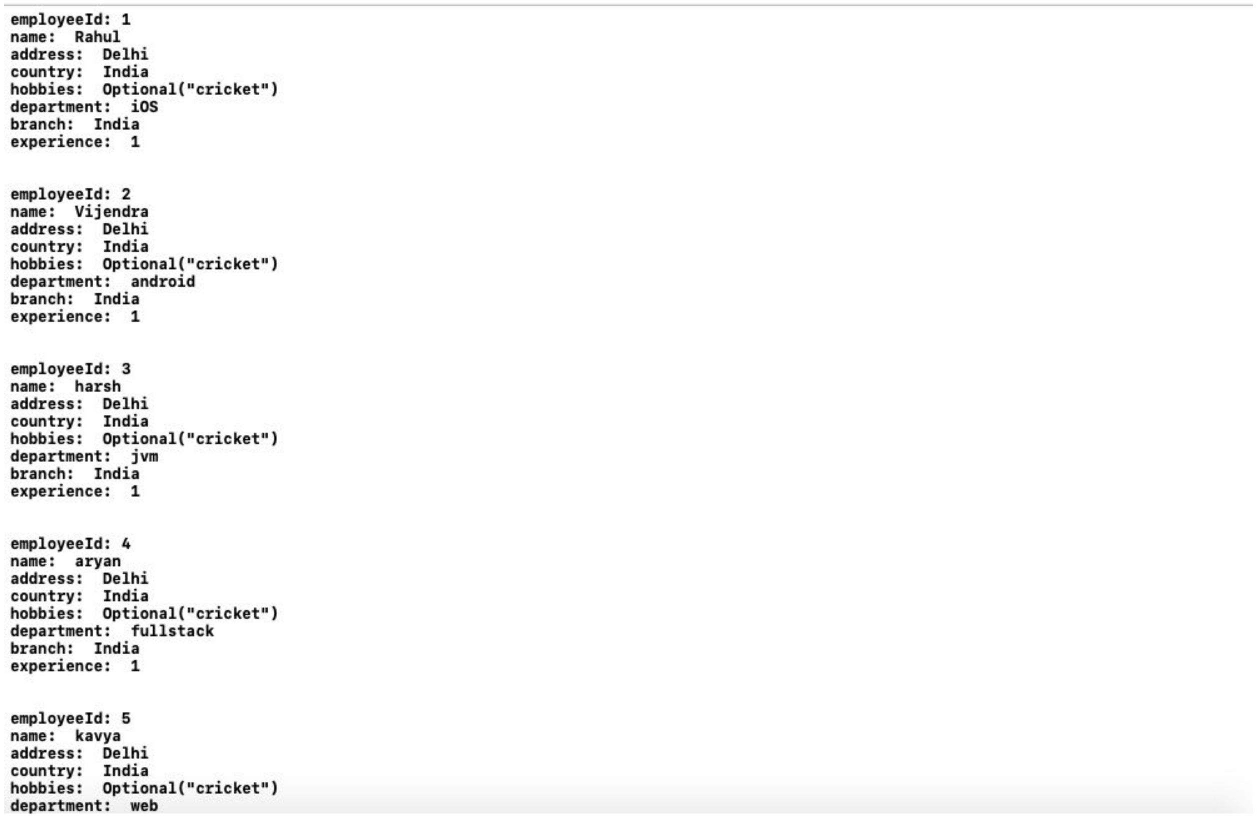
```swift
struct Employee {
    var id: Int
    var personalEmployee: EmployeePersonal
    var professionalEmployee: EmployeeProfessional

    init(ecid id: Int, personalEmployees: EmployeePersonal, professionalEmployees: EmployeeProfessional) {
        self.id = id
        self.personalEmployee = personalEmployees
        self.professionalEmployee = professionalEmployees
    }

    func displayInformation() {
        print("employeeId:", self.id)
        print("name: ", self.personalEmployee.name)
        print("address: ", self.personalEmployee.address)
        print("country: ", self.personalEmployee.country)
        print("hobbies: ", self.personalEmployee.hobbies as Any)
        print("department: ", self.professionalEmployee.department)
        print("branch: ", self.professionalEmployee.branch)
        print("experience: ", self.professionalEmployee.experience)
        print("\n")
    }
//    var e_P = employee_personal()
//    var e_Pro = employee_professional()
}
```



```swift
var employees: [Employee] = []

for item in zip(professionalEmployees, personalEmployees){
//    print(item.0.employeeId)
    if(item.0.employeeId == item.1.employeeId){
        employees.append(Employee(ecid: item.0.employeeId, personalEmployees: item.1,
            professionalEmployees: item.0))
    }
}
```

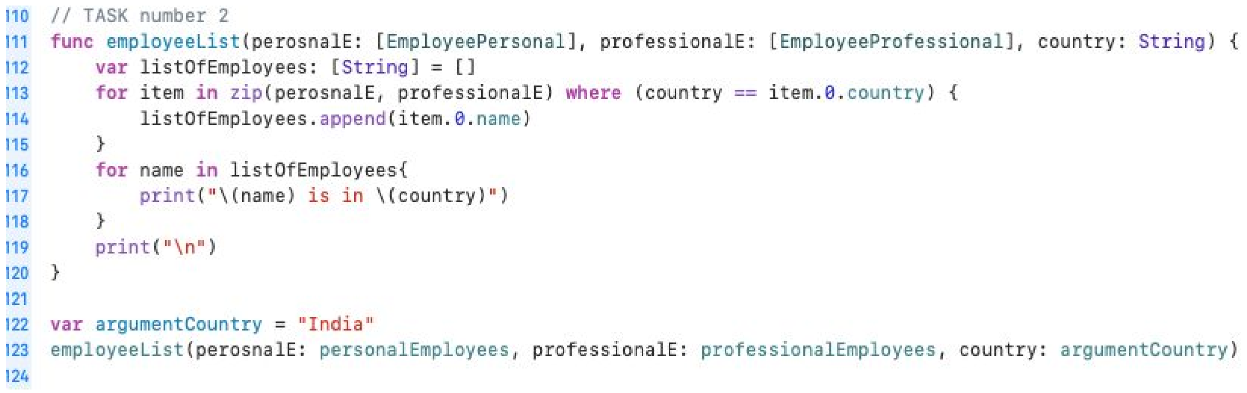


```swift
for employee in employees {
    employee.displayInformation()
}
```

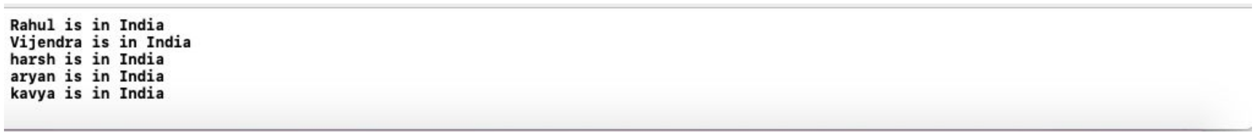


```
employeeId: 1
name:  Rahul
address:  Delhi
country:  India
hobbies:  Optional("cricket")
department:  iOS
branch:  India
experience:  1


employeeId: 2
name:  Vijendra
address:  Delhi
country:  India
hobbies:  Optional("cricket")
department:  android
branch:  India
experience:  1


employeeId: 3
name:  harsh
address:  Delhi
country:  India
hobbies:  Optional("cricket")
department:  jvm
branch:  India
experience:  1


employeeId: 4
name:  aryan
address:  Delhi
country:  India
hobbies:  Optional("cricket")
department:  fullstack
branch:  India
experience:  1


employeeId: 5
name:  kavya
address:  Delhi
country:  India
hobbies:  Optional("cricket")
department:  web
```

2. write a function that takes the two structure and give me list of all the employee that live in certain country

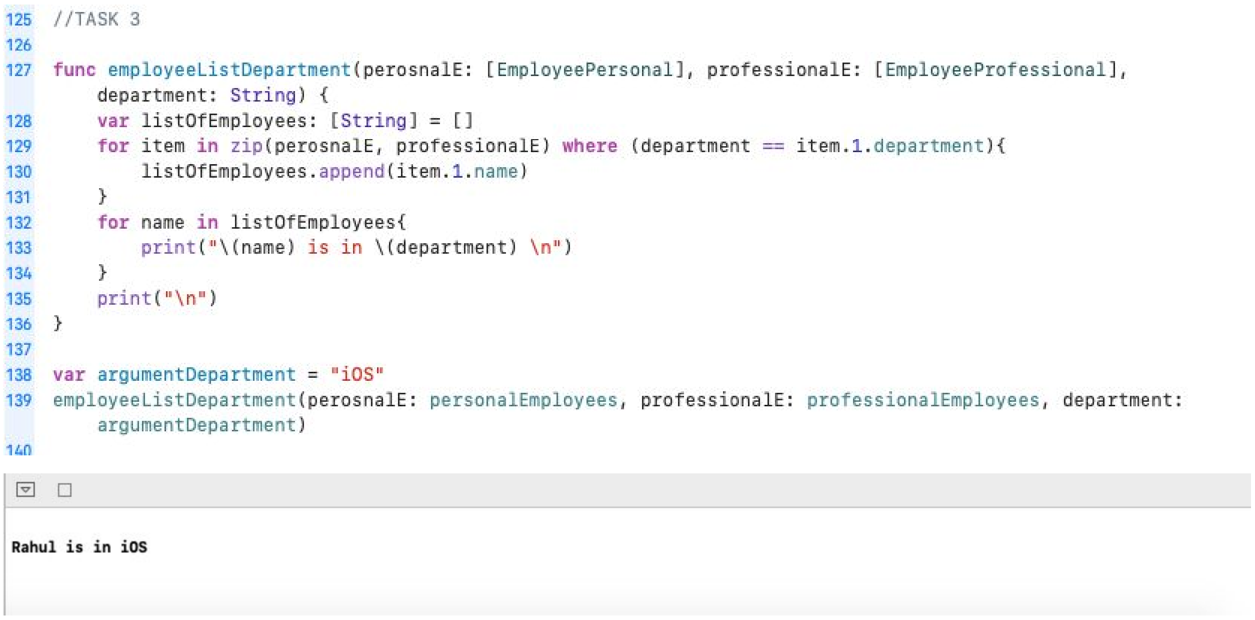


```swift
// TASK number 2
func employeeList(perosnalE: [EmployeePersonal], professionalE: [EmployeeProfessional], country: String) {
    var listOfEmployees: [String] = []
    for item in zip(perosnalE, professionalE) where (country == item.0.country) {
        listOfEmployees.append(item.0.name)
    }
    for name in listOfEmployees{
        print("\(name) is in \(country)")
    }
    print("\n")
}

var argumentCountry = "India"
employeeList(perosnalE: personalEmployees, professionalE: professionalEmployees, country: argumentCountry)
```



```
Rahul is in India
Vijendra is in India
harsh is in India
aryan is in India
kavya is in India
```

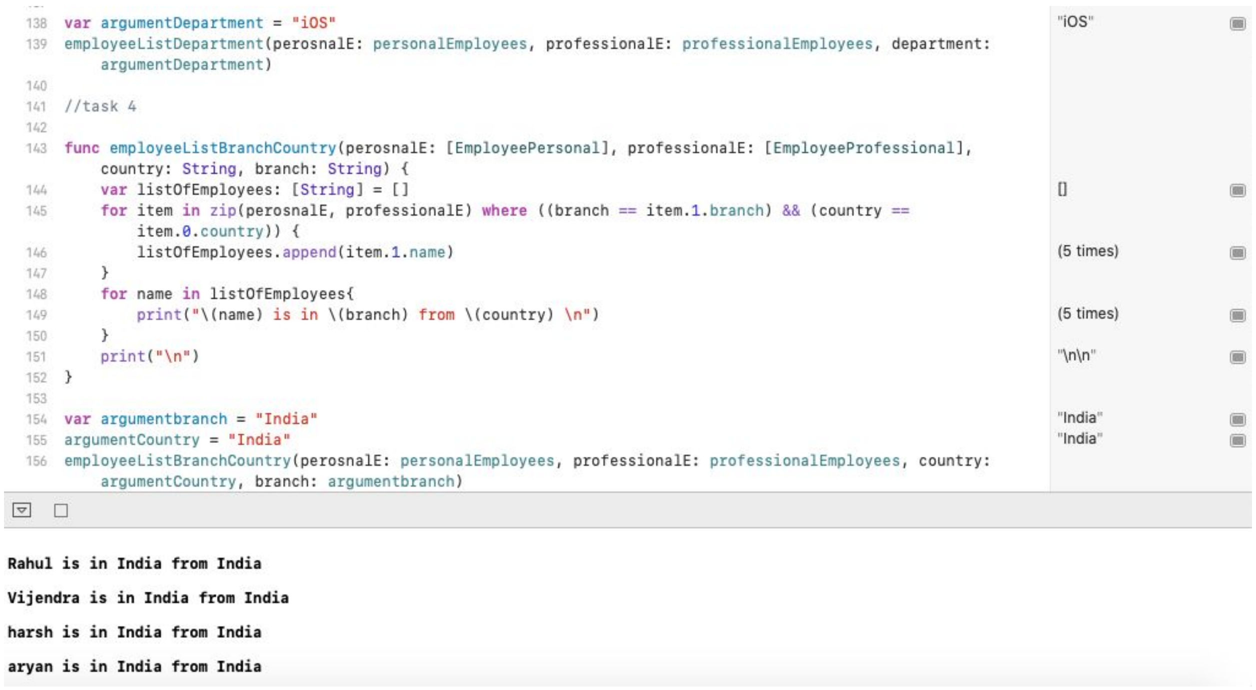
3. write a function that give me list of all the employee that live in certain department



```swift
//TASK 3

func employeeListDepartment(perosnalE: [EmployeePersonal], professionalE: [EmployeeProfessional],
    department: String) {
    var listOfEmployees: [String] = []
    for item in zip(perosnalE, professionalE) where (department == item.1.department){
        listOfEmployees.append(item.1.name)
    }
    for name in listOfEmployees{
        print("\(name) is in \(department) \n")
    }
    print("\n")
}

var argumentDepartment = "iOS"
employeeListDepartment(perosnalE: personalEmployees, professionalE: professionalEmployees, department:
    argumentDepartment)
```

```
Rahul is in iOS
```

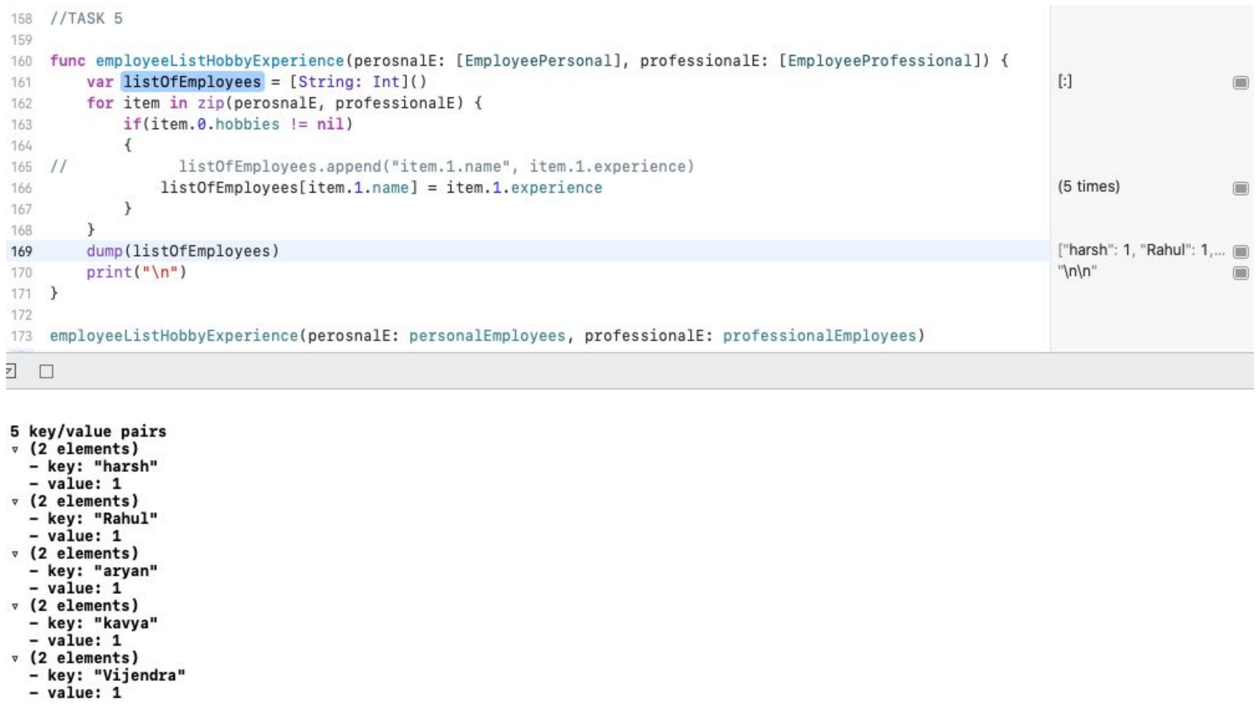
4. write a function that gives me a list of all the employees that live in the same country and work in the same branch.



```swift
var argumentDepartment = "iOS"
employeeListDepartment(perosnalE: personalEmployees, professionalE: professionalEmployees, department:
    argumentDepartment)

//task 4

func employeeListBranchCountry(perosnalE: [EmployeePersonal], professionalE: [EmployeeProfessional],
    country: String, branch: String) {
    var listOfEmployees: [String] = []
    for item in zip(perosnalE, professionalE) where ((branch == item.1.branch) && (country ==
        item.0.country)) {
        listOfEmployees.append(item.1.name)
    }
    for name in listOfEmployees{
        print("\(name) is in \(branch) from \(country) \n")
    }
    print("\n")
}

var argumentbranch = "India"
argumentCountry = "India"
employeeListBranchCountry(perosnalE: personalEmployees, professionalE: professionalEmployees, country:
    argumentCountry, branch: argumentbranch)
```

```
Rahul is in India from India
Vijendra is in India from India
harsh is in India from India
aryan is in India from India
```

5. write a function that returns me a list of all the employee names that has a hobby and with their experience.

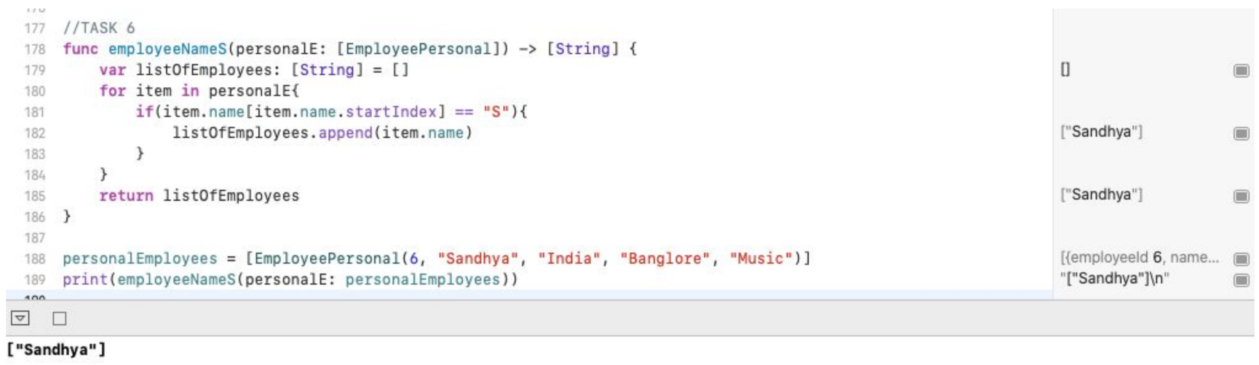


```swift
//TASK 5

func employeeListHobbyExperience(perosnalE: [EmployeePersonal], professionalE: [EmployeeProfessional]) {
    var listOfEmployees = [String: Int]()
    for item in zip(perosnalE, professionalE) {
        if(item.0.hobbies != nil)
        {
//            listOfEmployees.append("item.1.name", item.1.experience)
            listOfEmployees[item.1.name] = item.1.experience
        }
    }
    dump(listOfEmployees)
    print("\n")
}

employeeListHobbyExperience(perosnalE: personalEmployees, professionalE: professionalEmployees)
```

```
[:]
(5 times)
["harsh": 1, "Rahul": 1,...
"\n\n"
```

```
5 key/value pairs
▿ (2 elements)
  - key: "harsh"
  - value: 1
▿ (2 elements)
  - key: "Rahul"
  - value: 1
▿ (2 elements)
  - key: "aryan"
  - value: 1
▿ (2 elements)
  - key: "kavya"
  - value: 1
▿ (2 elements)
  - key: "Vijendra"
  - value: 1
```

6. write a function that return me list of all the employee name that starts with any “S”



```swift
//TASK 6
func employeeNameS(personalE: [EmployeePersonal]) -> [String] {
    var listOfEmployees: [String] = []
    for item in personalE{
        if(item.name[item.name.startIndex] == "S"){
            listOfEmployees.append(item.name)
        }
    }
    return listOfEmployees
}

personalEmployees = [EmployeePersonal(6, "Sandhya", "India", "Banglore", "Music")]
print(employeeNameS(personalE: personalEmployees))
```

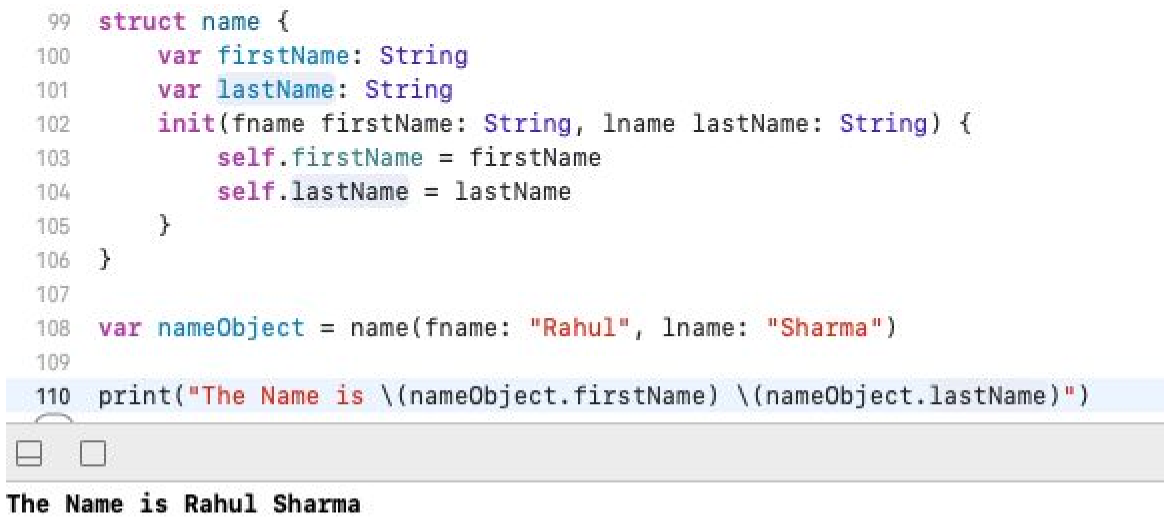
```
[]
["Sandhya"]
["Sandhya"]
[{employeeId 6, name...
"["Sandhya"]\n"
```

```
["Sandhya"]
```

Exercise 2

Initializers

Implement the parameterised initialisation with class or struct.



```swift
struct name {
    var firstName: String
    var lastName: String
    init(fname firstName: String, lname lastName: String) {
        self.firstName = firstName
        self.lastName = lastName
    }
}

var nameObject = name(fname: "Rahul", lname: "Sharma")

print("The Name is \(nameObject.firstName) \(nameObject.lastName)")
```

```
The Name is Rahul Sharma
```

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.

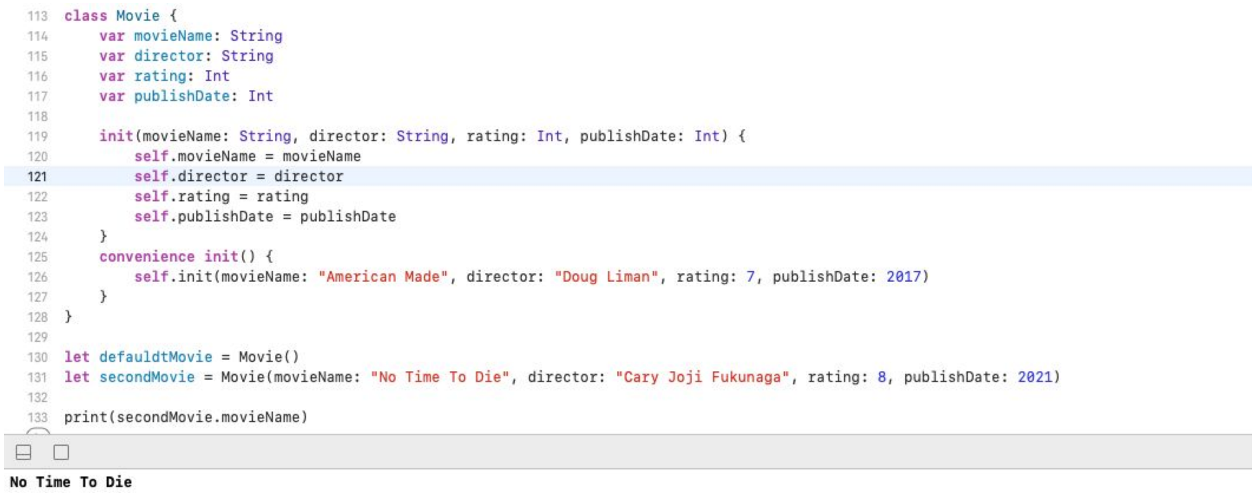
Rule 4: A designated initializer must ensure that all of the properties introduced by its class are initialized before it delegates up to a superclass initializer.

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.

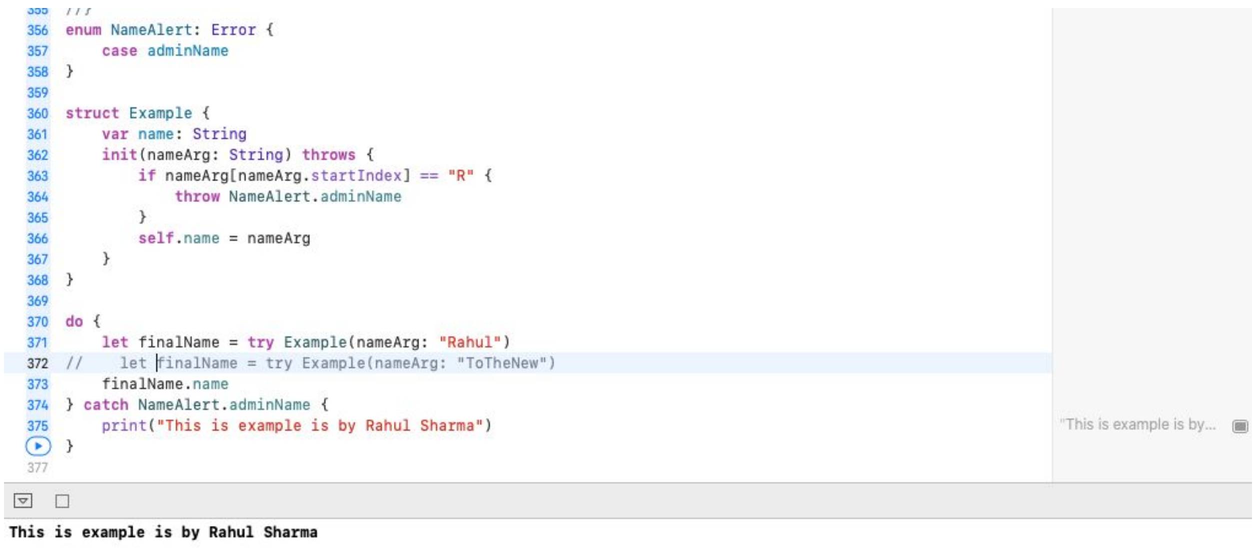
Using convenience Initializers, write-down the Initializers for MOVIE class having basic attributes like title, author, publish_date, etc.



```swift
class Movie {
    var movieName: String
    var director: String
    var rating: Int
    var publishDate: Int

    init(movieName: String, director: String, rating: Int, publishDate: Int) {
        self.movieName = movieName
        self.director = director
        self.rating = rating
        self.publishDate = publishDate
    }
    convenience init() {
        self.init(movieName: "American Made", director: "Doug Liman", rating: 7, publishDate: 2017)
    }
}

let defauldtMovie = Movie()
let secondMovie = Movie(movieName: "No Time To Die", director: "Cary Joji Fukunaga", rating: 8, publishDate: 2021)

print(secondMovie.movieName)
```

```
No Time To Die
```

Declare a structure which can demonstrate the throwable Initializer



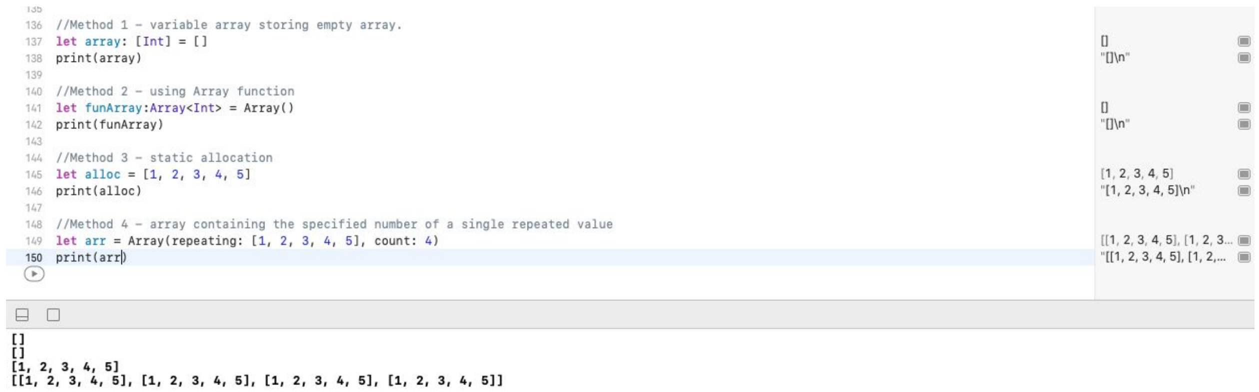
```swift
///
enum NameAlert: Error {
    case adminName
}

struct Example {
    var name: String
    init(nameArg: String) throws {
        if nameArg[nameArg.startIndex] == "R" {
            throw NameAlert.adminName
        }
        self.name = nameArg
    }
}

do {
    let finalName = try Example(nameArg: "Rahul")
//    let finalName = try Example(nameArg: "ToTheNew")
    finalName.name
} catch NameAlert.adminName {
    print("This is example is by Rahul Sharma")
}
```

"This is example is by...

```
This is example is by Rahul Sharma
```

Arrays

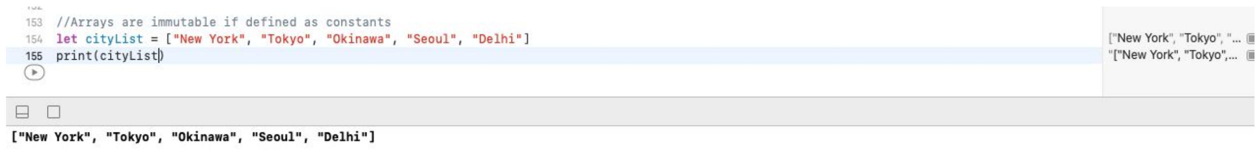
Create an array containing the 5 different integer values. Write at least 4 ways to do this.



```swift
//Method 1 - variable array storing empty array.
let array: [Int] = []
print(array)

//Method 2 - using Array function
let funArray:Array<Int> = Array()
print(funArray)

//Method 3 - static allocation
let alloc = [1, 2, 3, 4, 5]
print(alloc)

//Method 4 - array containing the specified number of a single repeated value
let arr = Array(repeating: [1, 2, 3, 4, 5], count: 4)
print(arr)
```

```
[]
[]
[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]]
```

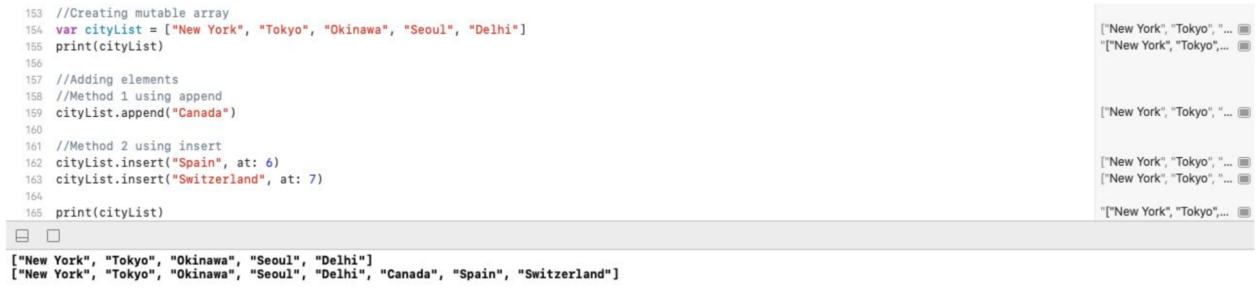
Create an immutable array containing 5 city names.



```swift
//Arrays are immutable if defined as constants
let cityList = ["New York", "Tokyo", "Okinawa", "Seoul", "Delhi"]
print(cityList)
```

```
["New York", "Tokyo", "Okinawa", "Seoul", "Delhi"]
```

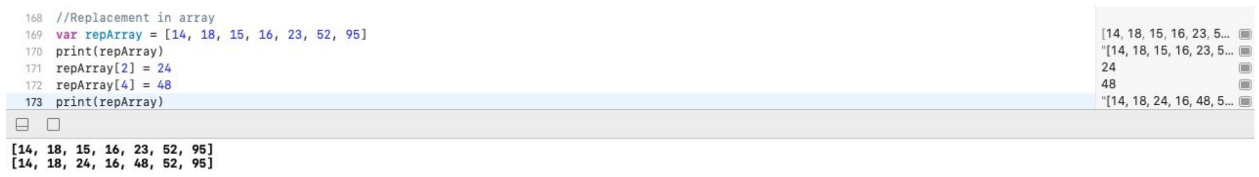
Create an array with city 5 city names. Later add other names like Canada, Switzerland, Spain to the end of the array in at least 2 possible ways.



```swift
//Creating mutable array
var cityList = ["New York", "Tokyo", "Okinawa", "Seoul", "Delhi"]
print(cityList)

//Adding elements
//Method 1 using append
cityList.append("Canada")

//Method 2 using insert
cityList.insert("Spain", at: 6)
cityList.insert("Switzerland", at: 7)

print(cityList)
```

```
["New York", "Tokyo", "Okinawa", "Seoul", "Delhi"]
["New York", "Tokyo", "Okinawa", "Seoul", "Delhi", "Canada", "Spain", "Switzerland"]
```

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



```swift
//Replacement in array
var repArray = [14, 18, 15, 16, 23, 52, 95]
print(repArray)
repArray[2] = 24
repArray[4] = 48
print(repArray)
```

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

Sets

1. Given the following sets:

let houseAnimals: Set = ["🐶", "🐱"]

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

let cityAnimals: Set = ["🐦", "🐭"]

Use set operations to...

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

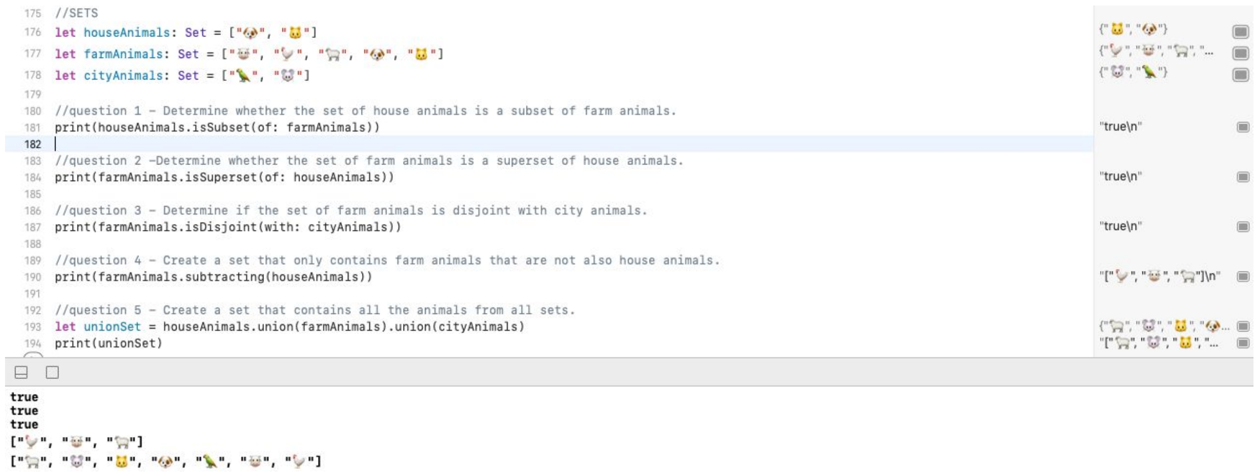
Determine whether the set of farm animals is a superset of house animals.

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

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

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

Answers of the following questions.



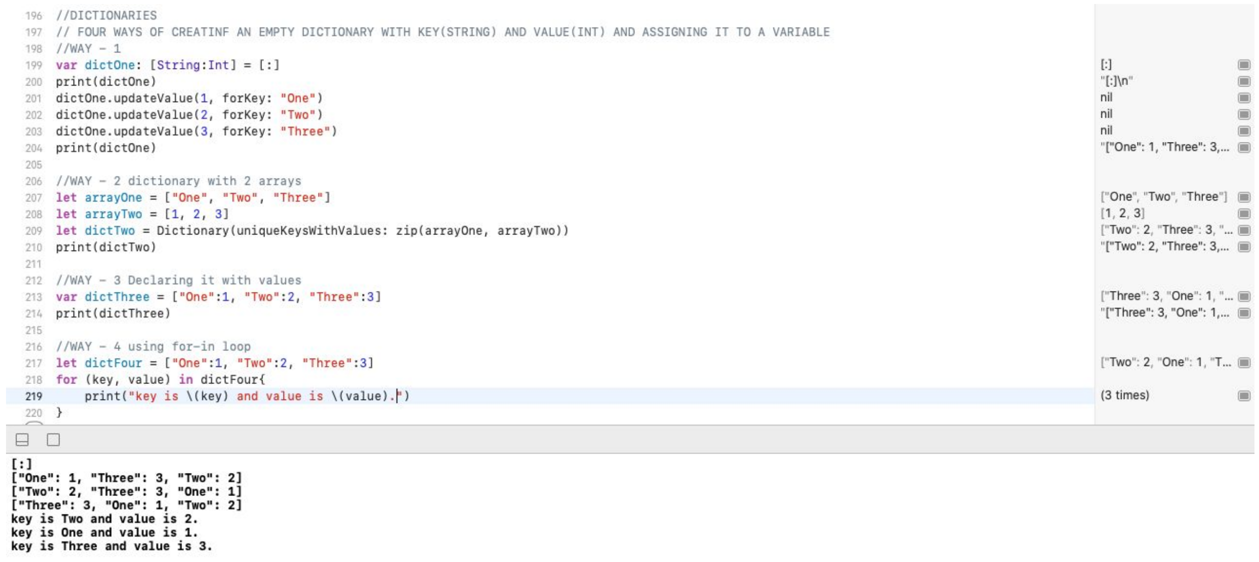
```
//SETS
let houseAnimals: Set = ["🐶", "🐱"]
let farmAnimals: Set = ["🐮", "🐔", "🐑", "🐶", "🐱"]
let cityAnimals: Set = ["🐦", "🐭"]

//question 1 - Determine whether the set of house animals is a subset of farm animals.
print(houseAnimals.isSubset(of: farmAnimals))

//question 2 -Determine whether the set of farm animals is a superset of house animals.
print(farmAnimals.isSuperset(of: houseAnimals))

//question 3 - Determine if the set of farm animals is disjoint with city animals.
print(farmAnimals.isDisjoint(with: cityAnimals))

//question 4 - Create a set that only contains farm animals that are not also house animals.
print(farmAnimals.subtracting(houseAnimals))

//question 5 - Create a set that contains all the animals from all sets.
let unionSet = houseAnimals.union(farmAnimals).union(cityAnimals)
print(unionSet)
```

```
{"🐱", "🐶"}
{"🐔", "🐮", "🐑", "...
{"🐭", "🐦"}
"true\n"
"true\n"
"true\n"
"["🐔", "🐮", "🐑"]\n"
{"🐑", "🐭", "🐱", "🐶...
"["🐑", "🐭", "🐱", "...
```

```
true
true
true
["🐔", "🐮", "🐑"]
["🐑", "🐭", "🐱", "🐶", "🐦", "🐮", "🐔"]
```

Dictionary

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).

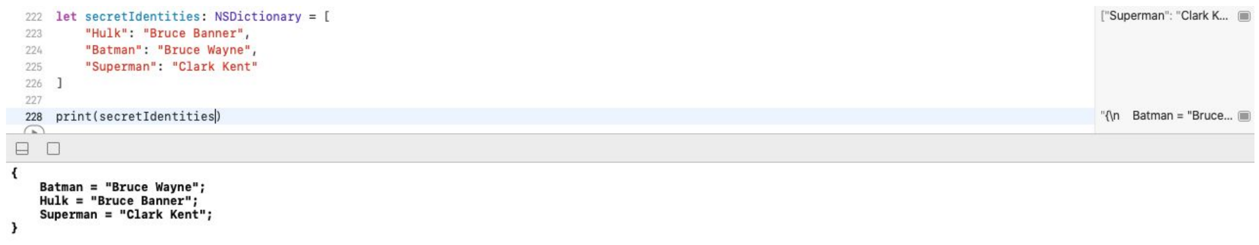


```swift
//DICTIONARIES
// FOUR WAYS OF CREATINF AN EMPTY DICTIONARY WITH KEY(STRING) AND VALUE(INT) AND ASSIGNING IT TO A VARIABLE
//WAY - 1
var dictOne: [String:Int] = [:]
print(dictOne)
dictOne.updateValue(1, forKey: "One")
dictOne.updateValue(2, forKey: "Two")
dictOne.updateValue(3, forKey: "Three")
print(dictOne)

//WAY - 2 dictionary with 2 arrays
let arrayOne = ["One", "Two", "Three"]
let arrayTwo = [1, 2, 3]
let dictTwo = Dictionary(uniqueKeysWithValues: zip(arrayOne, arrayTwo))
print(dictTwo)

//WAY - 3 Declaring it with values
var dictThree = ["One":1, "Two":2, "Three":3]
print(dictThree)

//WAY - 4 using for-in loop
let dictFour = ["One":1, "Two":2, "Three":3]
for (key, value) in dictFour{
    print("key is \(key) and value is \(value).")
}
```

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

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

["Three": 3, "One": 1, "...
"["Three": 3, "One": 1,...

["Two": 2, "One": 1, "T...
(3 times)
```

```
[:]
["One": 1, "Three": 3, "Two": 2]
["Two": 2, "Three": 3, "One": 1]
["Three": 3, "One": 1, "Two": 2]
key is Two and value is 2.
key is One and value is 1.
key is Three and value is 3.
```

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

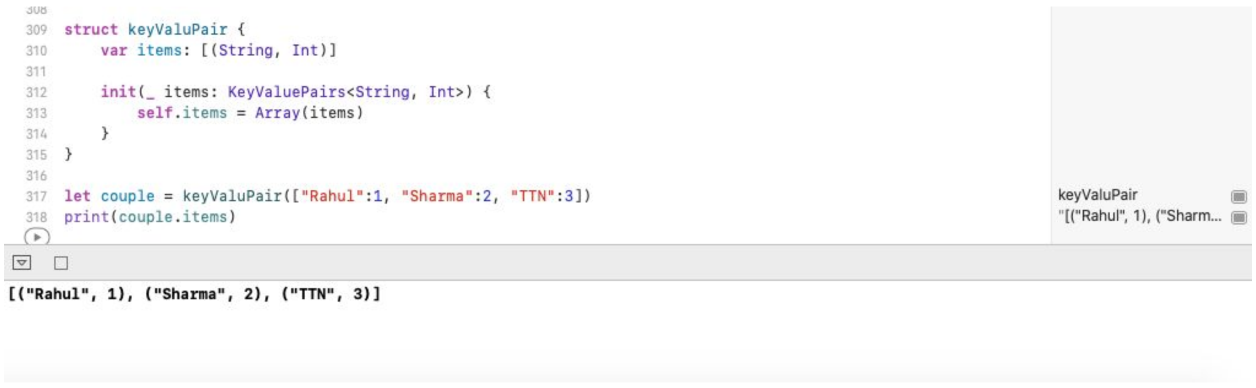


```swift
let secretIdentities: NSDictionary = [
    "Hulk": "Bruce Banner",
    "Batman": "Bruce Wayne",
    "Superman": "Clark Kent"
]

print(secretIdentities)
```

```
["Superman": "Clark K...
"{\n    Batman = "Bruce...
```

```
{
    Batman = "Bruce Wayne";
    Hulk = "Bruce Banner";
    Superman = "Clark Kent";
}
```

Create a nesters structure of Key-value pairs.

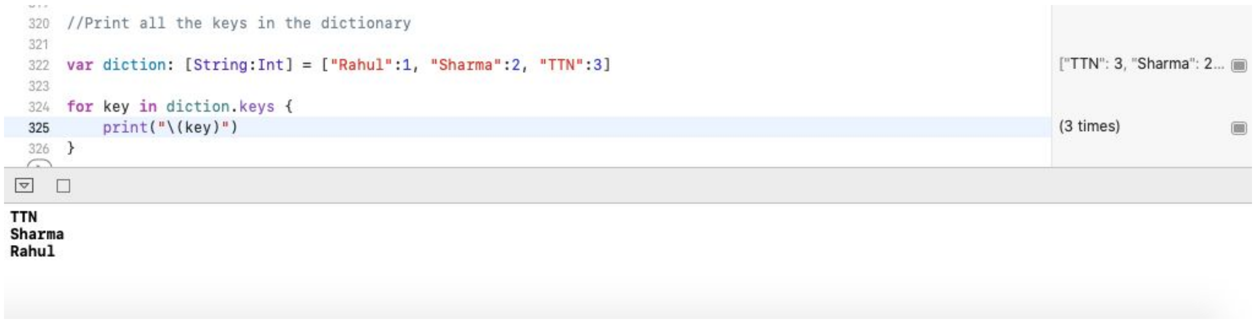


```
struct keyValuPair {
    var items: [(String, Int)]

    init(_ items: KeyValuePairs<String, Int>) {
        self.items = Array(items)
    }
}

let couple = keyValuPair(["Rahul":1, "Sharma":2, "TTN":3])
print(couple.items)
```

```
keyValuPair
"[("Rahul", 1), ("Sharm...
```

```
[("Rahul", 1), ("Sharma", 2), ("TTN", 3)]
```

Print all the keys in the dic



```
//Print all the keys in the dictionary

var diction: [String:Int] = ["Rahul":1, "Sharma":2, "TTN":3]

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

```
["TTN": 3, "Sharma": 2...
(3 times)
```

```
TTN
Sharma
Rahul
```

Subscript

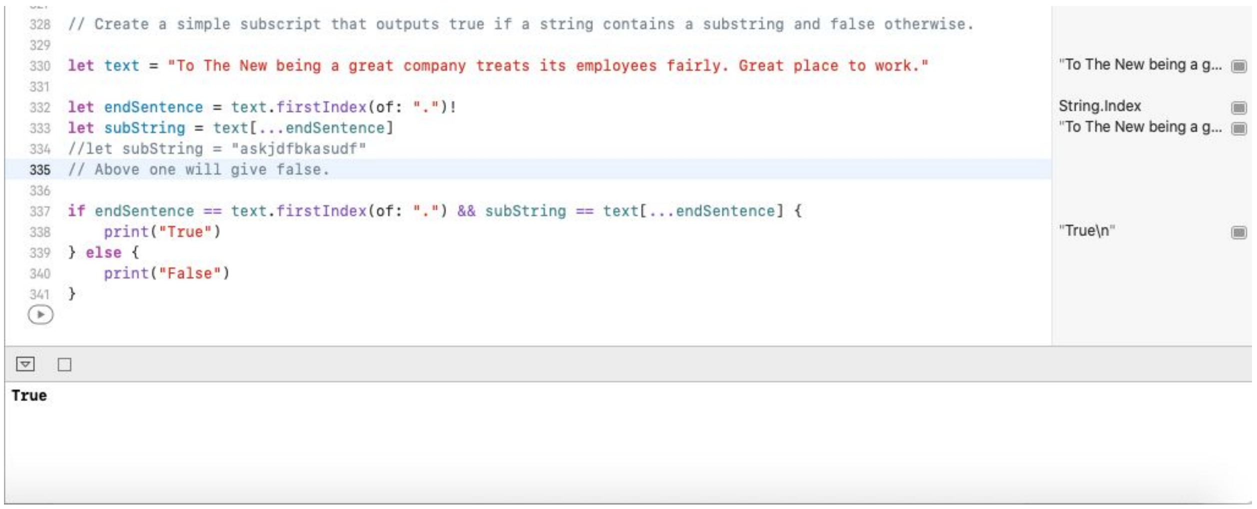
What is subscript ? Write down the declaration syntax.

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
 }
}
```

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



```swift
// Create a simple subscript that outputs true if a string contains a substring and false otherwise.

let text = "To The New being a great company treats its employees fairly. Great place to work."

let endSentence = text.firstIndex(of: ".")!
let subString = text[...endSentence]
//let subString = "askjdfbkasudf"
// Above one will give false.

if endSentence == text.firstIndex(of: ".") && subString == text[...endSentence] {
    print("True")
} else {
    print("False")
}
```

```
"To The New being a g...
String.Index
"To The New being a g...
"True\n"
```

```
True
```