Universitatea din Bucuresti
Facultatea de Matematica si Informatica

AppleLab @ UniBuc
Dezvoltare aplicatii iOS cu Swift 3

- CURS 2 -

Var, let, optionals, for-in
Classes, Inheritance

15.03.2017

iOS Course - 2

Noțiuni necesare pentru a continua dezvoltarea de aplicații sub Swift

- variabile/constante
- optionals (tip de date ce acoperă lipsa valorii variabilei)
- For-in: parcurgerea unui range (interval). *C++ deprecated*
- clasă (descriere / template pentru un obiect)
- instanță (cum se comportă o clasă)
- mesaj (trimis catre un obiect)
- metodă (codul invocat de un mesaj)
- instance variable (unde sunt stocate datele unui obiect)
- moștenire (superclasă - subclasă)

iOS Course - 2

VARIABILE/CONSTANTE

- TYPE INFERENCE

// compilatorul “deduce” că varA este Int
var varA = 42
// compilatorul “deduce” că varB este Double
var varB = 3.14159
// compilatorul “deduce” că varC este Double
var varC = 3 + 0.14159

- VARIABILE

varTest: Int = 4

- CONSTANTE

let constA = 42
let constB:Float = 3.14159

iOS Course - 2

OPTIONALS

- Tip de date OPTIONAL care poate avea două valori:
  - NONE sau NIL
  - SOME(T), unde T este valoarea asociată unui tip disponibil in Swift
- Declarare

```
var perhapsInt: Int?
```

- Utilizare

```
var myString: String? = nil
if myString != nil {
    print(myString)
} else {
    print("myString has nil value")
}
```

iOS Course - 2

OPTIONALS

Forced Unwrapping

Atunci când definim o variabilă ca fiind OPTIONAL, pentru a folosi valoarea din ea trebuie s-o “desfacem” - unwrap. Pentru asta, punem “!” după numele variabilei

```
var myString: String?
myString = "Hello Swift!"
if myString != nil {
   print(myString) // afisează Optional(“Hello Swift!”)
   print(myString!) // afisează"Hello Swift!”
}else{
   print("myString has nil value")
}
```

Tema nr.1: Studiu Optional Binding - Automatic Unwrapping

iOS Course - 2

For-in

```
var a = 2; var b = 120
for i in a...b { // intotdeauna a <= b
  print ( i )  // afiseaza toate numerele din intervalul [a, b]
}
```

b.)

```
var array: [Any] = ["mama", "are", 72, "mere", "eu", "am", 0]
for i in 3..<array.count { // i ∈ [3, (array.cout-1)]
  print ( array[i] )  // afiseaza mere, eu, am, 0 pe cate o linie
}
```

c.)

```
for i in array {
  print ( i )  // afiseaza toate elementele lui array pe cate o linie
}
```

iOS Course - 2

For-in

```
var wordsDictionary: [String: Any] = [
    "First prime number": 2,
    "iPhone": "Is best phone",
    "A Dictionary": "is like an array with any indexes"
]

for (key, value) in wordsDictionary { // intotdeauna a <= b
  print ( key + " is defined as: \( value ) ")
}

// Dictionary is defined as: is like an array with any indexes
// iPhone is defined as: Is the best phone
// First prime number is defined as: 2
```

Tema nr.2: Studiu din Swift Programming Language:

Array, Dictionary, Enum, Struct

iOS Course - 2

CLASE

BENEFICII

- **Prin moștenire proprietățile sunt “trecute” de la o clasă la alta**
- **Reference counting ARC (ce-o fi aia?) permite unei clase sa aibe cel puțin o instanță**
- **Proprietățile sunt definite pentru a stoca valori**
- **Metodele dezvoltă partea de funcționalitate**
- **Access level (seteaza tipul de acces asupra membrilor)**
- **Stările inițiale sunt setate prin initializer (init)**
- **Utilizarea de protocoale (ce-o fi aia?) extinde abilitățile unei clase**

Tema nr.3: Studiu Reference counting ARC si protocoale

iOS Course - 2

CLASE

EXEMPLU

```
class Human {
    var name = ""
    var height = 0.0 // stored property
    var description: [String: String] {
        return ["name": self.name, "height":"\(self.height)"]
    } // computed property
}

var person = Human() // instanță a clasei Om
person.name = "Mihai"
person.height = 1.73

print(person.description["name"]!)  // se  afisează Mihai
print(person.description["height"]!) // se afisează 1.73
```

iOS Course - 2

Instance Methods

- oferă acces la proprietățile definite în clasă
- oferă și definește funcționalitate pentru aceste proprietăți

```
class calculations {
   let a: Int // nu le-am atribuit valoare pentru că ele vor fi definite prin initializer
   let b: Int
   let res: Int
   init (a: Int, b: Int) {
       self.a = a
       self.b = b
       res = a + b }
    func tot (c: Int) -> Int {  // vom discuta mai târziu despre funcții
        return res - c
    }
    func printCurrentResult ()  {
        print("Result is: " + tot(c: 50) ) // afiseaza 850
        print("Result is: \( tot(c: 20) )") // afiseaza 820
} }

 let calculatedObject = calculations(a: 600, b: 300)
 calculatedObject.printCurrentResult() // accesarea unei metode se face prin “.”
```

iOS Course - 2

Instance Variables

- stored property - rețin o valoare constantă sau o variablă într-o instanță
- computed property - calculează valoarea și apoi o returneaza / rețin într-o instanță

```
class Sample {
  var no1 = 0.0, no2 = 0.0
  var length = 300.0, breadth = 150.0 // stored property
  var middle: (Double, Double) { // computed property
      get {
            return (length / 2, breadth / 2)
      }
      set(axis) {
            no1 = axis.0 - (length / 2)
            no2 = axis.1 - (breadth / 2)
      }
  }
}
var result = Sample()
print(result.middle) // afisează (150.0, 75.0)
result.middle = (0.0, 10.0)
print(result.no1) // afisează -150.0
print(result.no2) // afisează -65.0
```

iOS Course - 2

Moștenirea - Inheritance

- subclasă: când o clasă moștenește proprietăți și metode dintr-o altă clasă.
- superclasă: o clasă ce conține proprietăți și metode moștenite de o altă clasă.

```
class Rectangle { // superclasă
    var width = 42
    var height = 10
    var area: String { // computed property
        return "area \(self.width*self.height)px"
    }
}
class Square: Rectangle { // subclasa Square - moștenește clasa Rectangle
    var color = "Blue"
    override var area: String { // proprietate supra-scrisă
        return super.area + " override a rectangle and its color is \(self.color)"
    }
}

let squ = Square()
squ.width = 10
squ.color = "Pink"
print("Square with \(squ.area)") // Square with area 100px override a rectangle and its
color is Pink
```

iOS Course - 2

Access levels

- private: disponibil doar la nivel de obiect, nu este accesibil la exterior si nici mostenit
- public: este accesibil atat la nivel de obiect, de subclase cat si din exterior

```
class Rectangle {
    var width = 42; var height = 10
    private var angle = 90
    private var area: String {
        return "area \(self.width*self.height)px"
    }
    public func getArea() -> String { return self.area }
}
class Square: Rectangle {
    var color = "Blue"
    override var area: String { // Eroare. area este privat, nu pot face override
        return super.getArea() + " override a rectangle and its color is \(self.color)"
                // super.area nu ar fi functionat, area este privat in clasa Rectangle
    }
}

let squ = Square()
squ.width = 10
print("Square with \(squ.area)") // Fara override: Succes. area este public si definit in
clasa Square, nu este mostenit din Rectangle
squ.angle = 30 // Eroare. angle este privat in clasa Rectangle
```

Demo

Continuare aplicatie ***NrPrim***

Playgrounds:

https://github.com/mariusjcb/iOS10_PLAYGROUNDS_CS2

Proiect Xcode:

https://github.com/mariusjcb/iOS10_CS2