7.0 Lesson - Swift 2 Basics

Paul Solt - Paul@SuperEasyApps.com
SuperEasyApps.com

7.1 Lecture - Variables and Types

Data can be different kinds of information like: numbers, text, video, images, animations, and database files. In your first iPhone apps you will need to store both text and numeric data types, which you will learn about in this lesson.

Links

- Swift Programming Language Basics

7.2 Tutorial - Variables using the var and let Keywords

There are two keywords for creating new variables: `var` and `let`.

1. `var` creates variables that can change values

```
var speed = 60.0 // mph
speed = 75.0
speed = 95.0
```

2. `let` creates constant variables that cannot change values

```
let speedLimit = 65.0 // mph
speedLimit = 200.0 // Error
let speedingTicket = speed - speedLimit
```

Links

- Swift Programming Language Basics

7.3 Tutorial - Numbers in Swift

There are two primary types of numbers that you will work with in Swift: whole numbers and numbers with a decimal place. Computers store these types of numbers differently, so there is a separation in code.

1. Integers (Whole numbers)

```
let stepsToday: Int = 4580
let stepsLastWeek: Int = 54084
let averageStepsPerDay = stepsLastWeek / 7  // 7 days

var age = 29
age = age + 1
```

2. Floating-point numbers (Real numbers)

```
let pi_less_accurate: Float = 3.14
let pi: Double = 3.1415926
let radius = 10 // car wheel
let area = pi * radius * radius // PI * R^2
```

3. Swift is strongly typed, so you have to be very explicit when adding different types of numbers together. This means you need to convert between types using initializers for the type of data. For `Int` use: `Int(3.5)` and for `Double` use: `Double(45)`

```
let x: Int = 25
let userInput: Double = 500.75
// let sum = x + userInput  // ERROR!
let sum = Double(x) + userInput  // Convert Int using Double initializer
```

Links

- Swift Programming Language Basics

7.4 Tutorial - String and Character Types

1. Characters

```
let letter: Character = "a"
let word: String = "apple"
let lastLetter = word.characters.last!
```

2. Adding `String` values together

```
let firstName = "Paul"
let lastName = "Solt"
let fullName = firstName + " " + lastName
```

Links

- Strings and Characters - apple.com
- Control Flow - Conditional Statements

7.5 Lecture - Conditionals and if Statements

Conditional expressions along with if, if/else, and if/else if/else structures enable you to create logic that powers your iPhone apps.

Links

- Control Flow - Conditional Statements

7.6 Tutorial - Conditionals and if Statements

To create logic for your apps, you will use conditional statements that can be evaluated as `true` or `false`. These statements can be used with the `if`statement to control the flow of code in your app. `if` statements give your app the ability to make a decision based on the result of a conditional statement.

1. Equality comparisons

```
var firstLetter: Character = "P"
firstLetter == "d"
firstLetter == "P"
firstLetter != "z"
```

2. Numeric comparisons

```
let x = 27
x > 25
x < 50
x > 2000
x == 27

let cost = 5.50
cost >= 5.50
cost <= 10.00
```

3. `if` statements

```
let todaysTemperature = 20 // Fahrenheit
if todaysTemperature <= 32 {
    print("It's freezing, grab a heavy jacket!")
}
```

Links

- Control Flow - Conditional Statements

7.7 Tutorial - Logical Operators and if else Statements

To create more complex logic you will want to combine statements using logical operators. Combining statements together makes your apps more expressive and concise. You can trigger logic to happen if only two or more conditions are true, or if any of the given conditions are true.

1. Logical operators (compound expressions)
    1. And: `&&`

```
let temperature = 20 // Fahrenheit
if temperature > 50 && temperature < 65 {
    print("It's chilly wear a hoody!")
}
```

2. Or: `||`

```
let totalApples = 20
let totalOranges = 5

if totalApples > 20 || totalOranges > 20 {
    print("Sell some fruit at the fruit stand")
}
```

2. `if/else` statements

```
var hour = 9 // 24 hour time
var name = "Paul"

if hour >= 4 && hour < 12 {
    print("Good morning \(name)")
} else if hour >= 12 && hour < 17 {
    print("Good afternoon \(name)")
} else if hour >= 17 && hour < 24 {
    print("Good evening \(name)")
} else {
    print("Enjoy your day \(name)")
}
```

Links

- Control Flow - Conditional Statements

7.8 Tutorial - Optional Types, nil, and Optional Binding

Optional types are an addition to Swift that allow you to express if values are present or absent. Many times when you are working with user interfaces, user input, unreliable data, and server requests your app may be in a state where the data doesn't exist (yet!). Optionals give your app the ability to represent when data may or may not be present using the `nil` keyword or the value.

1. Non-optional types look like: `Int`

2. Optional types have a `?` at the end of the type: `Int?`
3. Optional types can be a value or have no value: `nil`.
4. Optional Types

```
var optionalName: String? = nil
optionalName = "Paul"
// force unwrap
print("name:" + optionalName!)
```

5. User Input example

```
var userInputString = "100" // "word"
var optionalInt = Int(userInputString)
if optionalInt != nil {  // Valid value
    var number = optionalInt!       // Force unwrap
    print("The number is valid: \(number)")
} else {  // Invalid value
    print("Invalid number: \(userInputString)")
}
```

6. Optional Binding (i.e.: less code to write)

```
if let number = Int(userInputString) {
    print("The number is valid: \(number)")
} else {
    print("Invalid number: \(userInputString)")
}
```

Links

- Optional Types - Swift Basics - apple.com