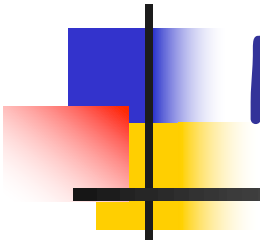


Introduction to iOS Programming and Swift

Dr. Eyuphan Bulut

VCU
School of Engineering | Computer Science

iOS Apps

2 million iOS apps…. (as of June/16)

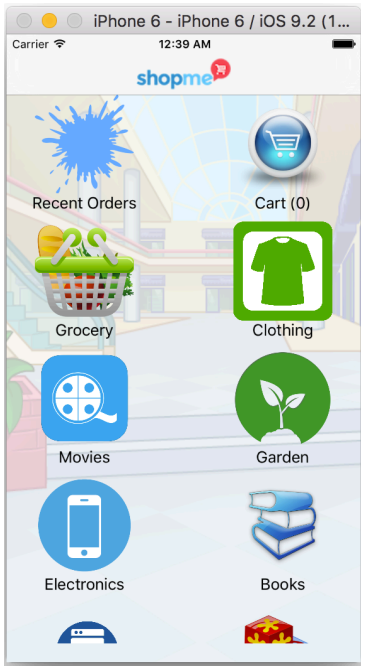
150 billion downloads

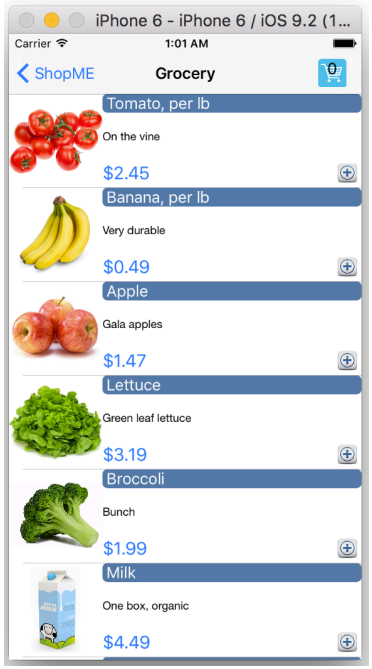
https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

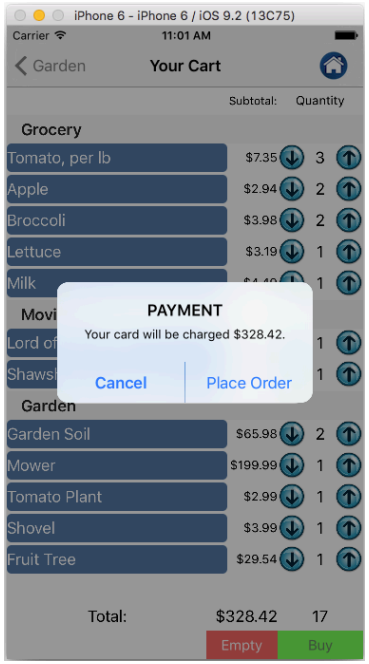
Student Apps

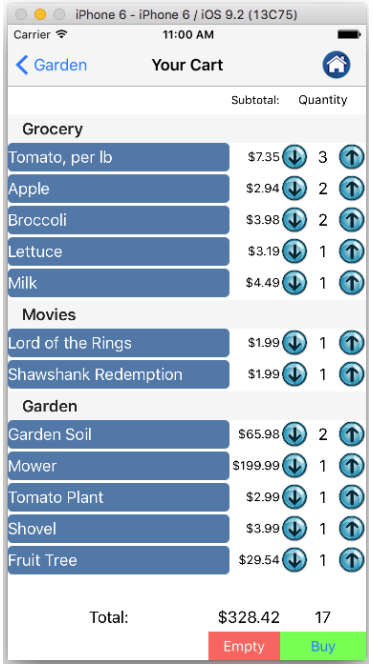
Some of the apps designed in CMSC 491 Mobile iOS Programming course in Spring 2016

- GPA Calculator
- ShopMe App



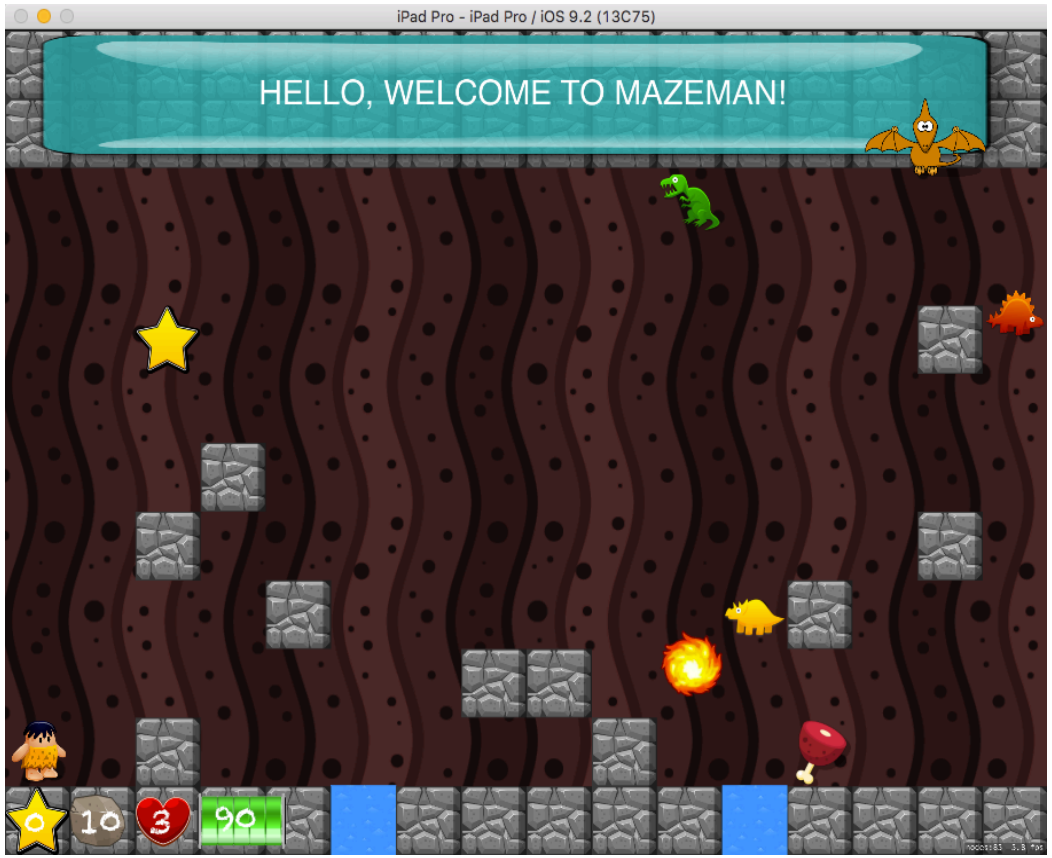


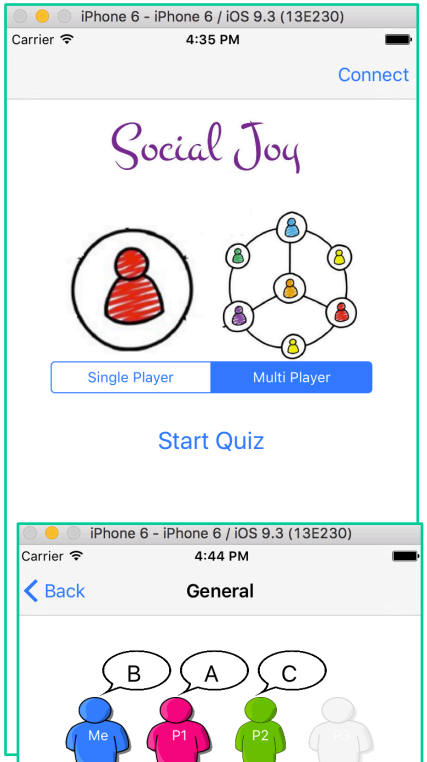


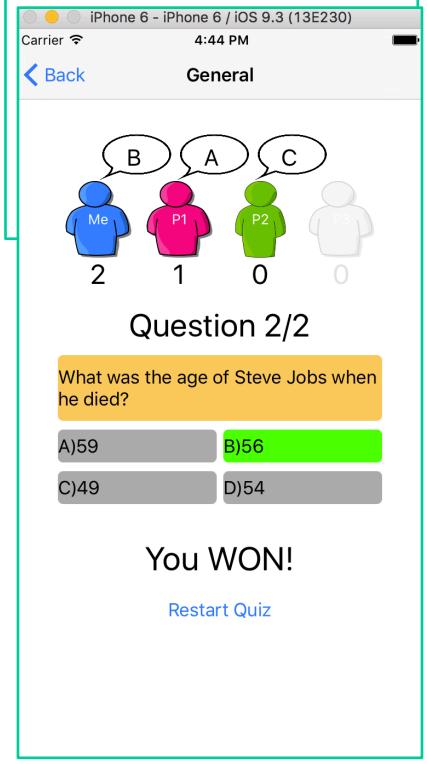


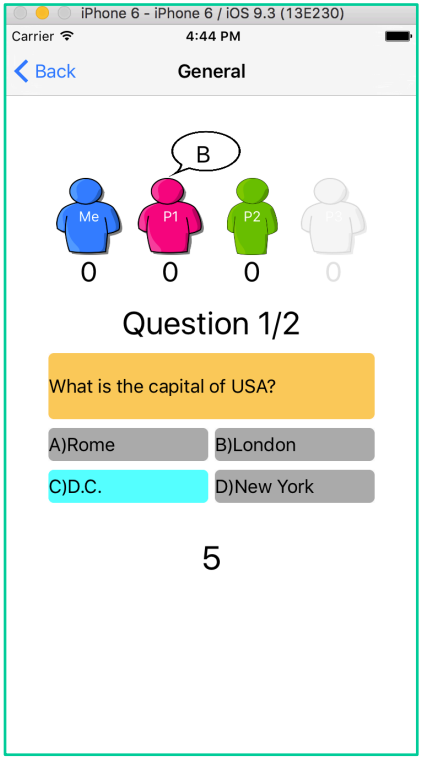
Student Apps

- Games
- Multi-player Quiz game









iPhone generations

Billionth iPhone was sold on June 27, 2016.

Image src: http://betanews.com/2015/09/30/iphone-supertest-all-12-of-apples-iphones-go-head-to-head-but-which-one-wins/

iOS Application Development

- Tools

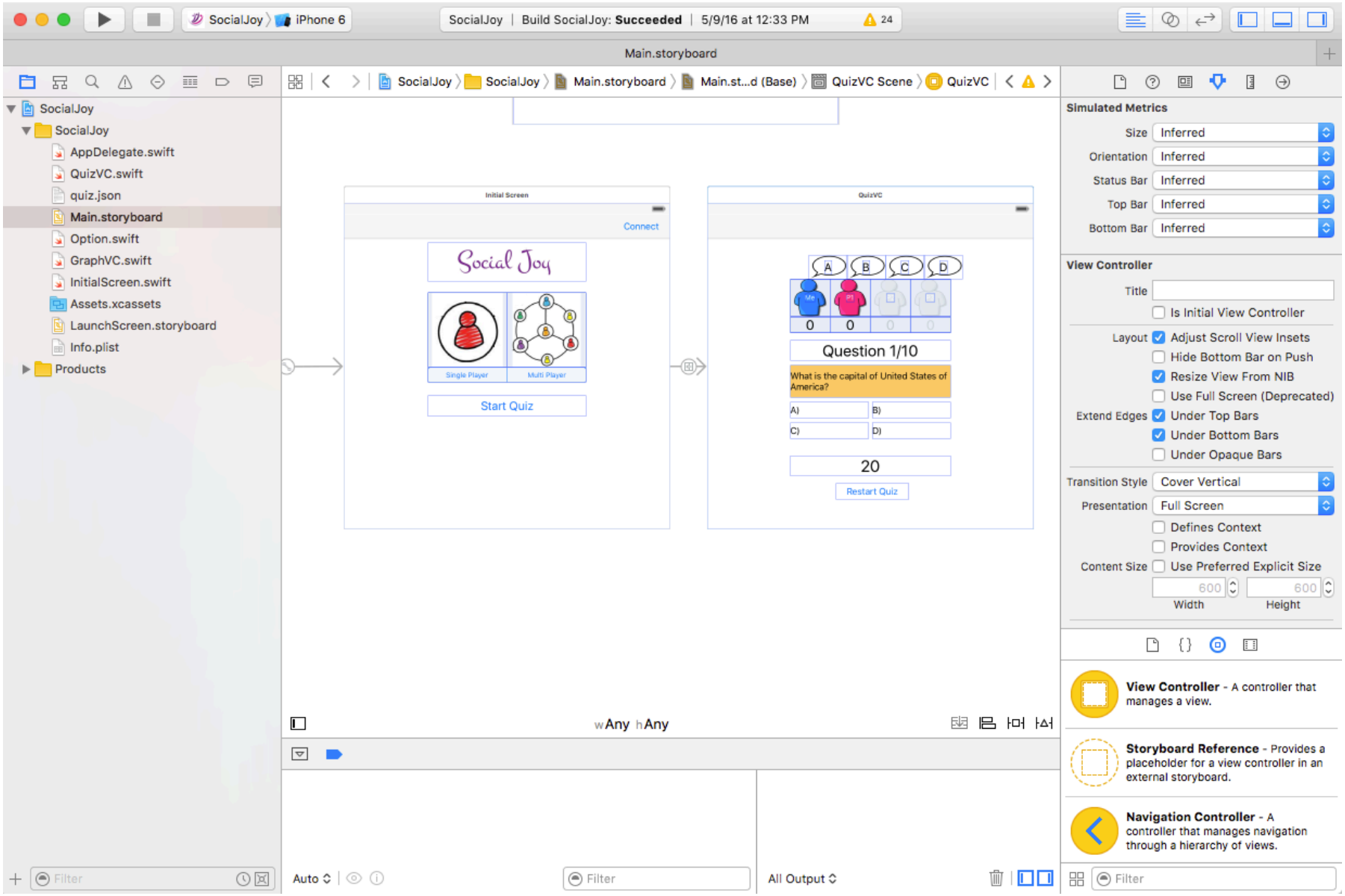
iOS

Xcode 8.1

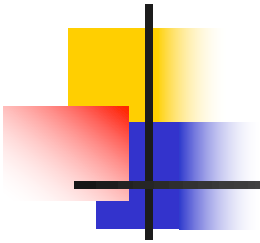
Swift 3

https://developer.apple.com/

Xcode IDE


SocialJoy
iPhone 6
SocialJoy | Build SocialJoy: Succeeded | 5/9/16 at 12:33 PM
24
Main.storyboard
SocialJoy
SocialJoy
AppDelegate.swift
QuizVC.swift
quiz.json
Main.storyboard
Option.swift
GraphVC.swift
InitialScreen.swift
Assets.xcassets
LaunchScreen.storyboard
Info.plist
Products
Initial Screen
Connect
Social Joy
Single Player
Multi Player
Start Quiz
QuizVC
A
B
C
D
Me
P1
0
0
Question 1/10
What is the capital of United States of America?
A)
B)
C)
D)
20
Restart Quiz
wAny hAny
Simulated Metrics
Size
Inferred
Orientation
Inferred
Status Bar
Inferred
Top Bar
Inferred
Bottom Bar
Inferred
View Controller
Title
Is Initial View Controller
Layout
Adjust Scroll View Insets
Hide Bottom Bar on Push
Resize View From NIB
Use Full Screen (Deprecated)
Extend Edges
Under Top Bars
Under Bottom Bars
Under Opaque Bars
Transition Style
Cover Vertical
Presentation
Full Screen
Defines Context
Provides Context
Content Size
Use Preferred Explicit Size
600
600
Width
Height
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storyboard.
Navigation Controller - A controller that manages navigation through a hierarchy of views.
Auto
All Output
Filter

iOS Frameworks



Visual

Audio/Video

AVKit, CoreVideo, CoreAudio

Animation/ Graphics/Image

SprikeKit, SceneKit, UIKit, CoreImage, GameKit

User

User Accounts

AddressBook, Accounts, Social, EventKit

Email/Text/Call

CoreText, **MessageUI**, CoreTelephony

Data

CoreData, HealthKit, **Foundation** CloudKit

System

System/IO

IOKit, **Notification Center**

Security

Security Kit, LocalAuthentication

Connectivity

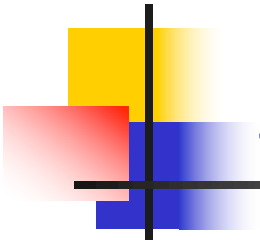
HomeKit, **MultiPeer Connectivity, CoreBluetooth**

Sensors/ Location

CoreLocation, MapKit, CoreMotion

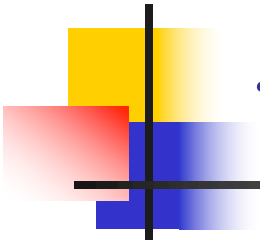
Advertising and Purchasing

AdSupport, Store Kit, iAd



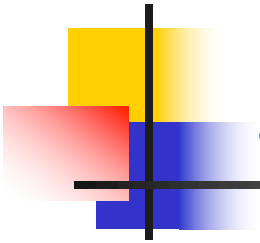
Swift

- Apple Swift Guide
  - https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/
- iOS Developer Program - $99/year
- iOS Developer University Program –free
  - Apps can be shared within students registered.



The Swift Programming Language

- New programming language for iOS, macOS, watchOS, and tvOS apps
- Builds on the best of C and Objective-C
- More flexible, readable and make programming easier
- Object-oriented
- Supports playgrounds, where you can see immediate output of your code



Swift Editors

RunSwift

http://www.runswiftlang.com/

IBM Swift Sandbox

https://swiftlang.ng.bluemix.net/#/repl

Swift Variables

Constants and Variables

Every variable and constant must be initialized before used.

No semicolons needed at the end

```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

Type Inference

Other types: Float, Double, Bool, UInt

```
let implicitInteger = 70
let implicitDouble = 70.0
let explicitDouble: Double = 70
```

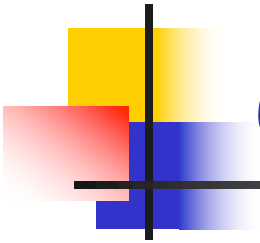
Conversion to different type

No implicit conversion between types

```
let label = "The width is "
let width = 94
let widthLabel = label + String(width)
```

Simpler way of having values in Strings

```
let apples = 3
let oranges = 5
let appleSummary = "I have \(apples) apples."
let fruitSummary = "I have \(apples + oranges) pieces of fruit."
```



Optionals

Optionals (?) may not have value

```
let optionalInt: Int? = 9
```

Force unwrap, if surely it has value

Optionals say either "there is a value, and it equals x" or "there isn't a value at all".

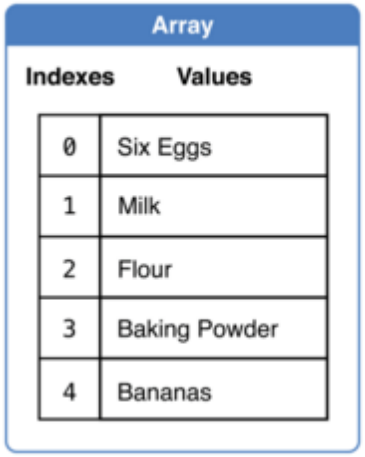
```
let actualInt: Int = optionalInt!
```

Where optionals are useful

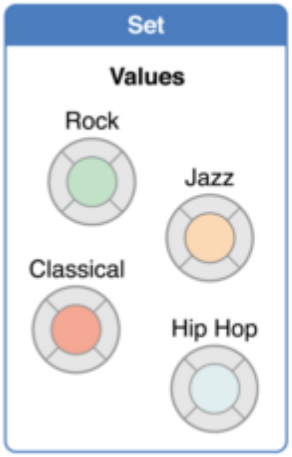
```
var myString = "7"
var possibleInt = Int(myString)
print(possibleInt)
```

```
myString = "banana"
possibleInt = Int(myString)
print(possibleInt)
```

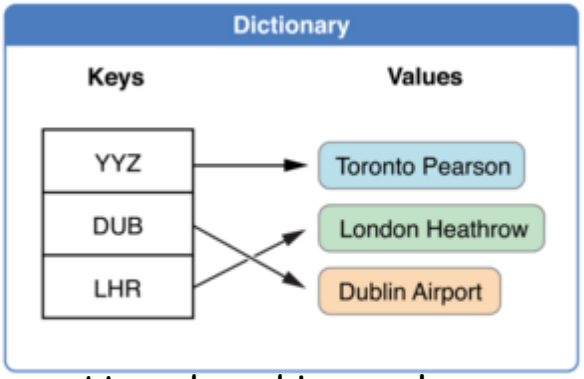
Collection Types



Ordered values



Unordered unique values



Unordered key-value

Arrays

Start index is 0
Default value array

.isEmpty, .append,
.insert, .removeAtIndex()
range

// enumerating an Array
for i in ratingList {
}

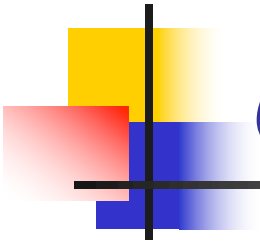
```
var ratingList = ["Poor", "Fine", "Good", "Excellent"]
ratingList[1] = "OK"
ratingList

// Creates an empty array.
let emptyArray = [String]()
Array<Element>

var threeDoubles = [Double](count: 3, repeatedValue: 0.0)

shoppingList[4...6] = ["Bananas", "Apples"]
```

same as Array<String>()



Control Flows

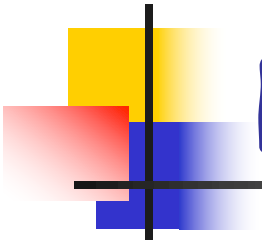
If statement

No parenthesis needed

```
let number = 23
if number < 10 {
    print("The number is small")
} else if number > 100 {
    print("The number is pretty big")
} else {
    print("The number is between 10 and 100")
}
```

For in

```
let individualScores = [75, 43, 103, 87, 12]
var teamScore = 0
for score in individualScores {
    if score > 50 {
        teamScore += 3
    } else {
        teamScore += 1
    }
}
print(teamScore)
```



Range

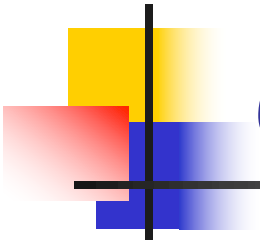
Half-open range operator (..<)

```
var firstForLoop = 0
for i in 0..<4 {
    firstForLoop += i
}
print(firstForLoop)
```

Closed range operator (...)

```
var secondForLoop = 0
for _ in 0...4 {
    secondForLoop += 1
}
print(secondForLoop)
```

Underscore(_) is wildcard. Used when iteration info is not needed



Optional binding

To check whether optional has value

```
let possibleNumber = "123"
let convertedNumber = Int(possibleNumber)
if convertedNumber != nil {
    print("convertedNumber has an integer value of \(convertedNumber!).")
}
```

```
if let actualNumber = Int(possibleNumber) {
    print("\"\(possibleNumber)\" has an integer value of \(actualNumber)")
} else {
    print("\"\(possibleNumber)\" could not be converted to an integer")
}
```

Where clause

```
var optionalHello: String? = "Hello"
if let hello = optionalHello where hello.hasPrefix("H"), let name = optionalName {
    greeting = "\(hello), \(name)"
}
```

Function Basics

Definition

```
func greet(person: String) -> String {
    let greeting = "Hello, " + person + "!"
    return greeting
}
```

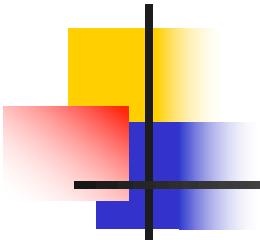
Usage

```
print(greet(person: "Anna"))
// Prints "Hello, Anna!"
print(greet(person: "Brian"))
// Prints "Hello, Brian!"
```

Method is function defined only for specific type

```
let exampleString = "hello"
if exampleString.hasSuffix("lo") {
    print("ends in lo")
}
```

```
var array = ["apple", "banana", "dragonfruit"]
array.insert("cherry", atIndex: 2)
array
```



Function Basics

Each function has a type, consisting of function's parameter types and return type

This allows passing functions to other functions

Parameters are not necessary

Default value

```
func greetAgain(person: String) -> String {
    return "Hello again, " + person + "!"
}
print(greetAgain(person: "Anna"))
// Prints "Hello again, Anna!"
```

Its type is (String)->String

```
func sayHelloWorld() -> String {
```

```
func greet(person: String) {
```

```
func someFunction(parameterWithoutDefault: Int, parameterWithDefault: Int = 12) {
```

Functions returning Tuples

Multiple value return as a tuple

Optional tuple is possible

(Int, Int)? Vs.

(Int?, Int?)

```
func minMax(array: [Int]) -> (min: Int, max: Int) {
    var currentMin = array[0]
    var currentMax = array[0]
    for value in array[1..<array.count] {
        if value < currentMin {
            currentMin = value
        } else if value > currentMax {
            currentMax = value
        }
    }
    return (currentMin, currentMax)
}
let bounds = minMax([8, -6, 2, 109, 3, 71])
print("min is \(bounds.min) and max is \(bounds.max)")
// prints "min is -6 and max is 109"
```

Argument Label and Parameter Name

Each function has:

-Argument Label

-Parameter Name

Argument Label is used when calling the function

Parameter name is used in the implementation of the function

```
func someFunction(firstParameterName: Int, secondParameterName: Int) {
    // In the function body, firstParameterName and secondParameterName
    // refer to the argument values for the first and second parameters.
}
someFunction(firstParameterName: 1, secondParameterName: 2)
```

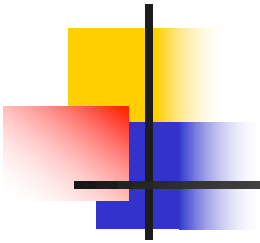
```
func someFunction(argumentLabel parameterName: Int) {
```

```
func greet(person: String, from hometown: String) -> String {
    return "Hello \(person)!  Glad you could visit from \(hometown)."
}
print(greet(person: "Bill", from: "Cupertino"))
```

By default, parameters use their parameter name as their argument label.

Argument label can be omitted by _

```
func someFunction(_ firstParameterName: Int, secondParameterName: Int) {
    // In the function body, firstParameterName and secondParameterName
    // refer to the argument values for the first and second parameters.
}
someFunction(1, secondParameterName: 2)
```



Function Types

Function Types:

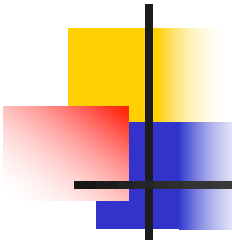
Both have type
(Int,Int)-> Int

```
func addTwoInts(_ a: Int, _ b: Int) -> Int {
    return a + b
}
func multiplyTwoInts(_ a: Int, _ b: Int) -> Int {
    return a * b
}
```

```
var mathFunction: (Int, Int) -> Int = addTwoInts

print("Result: \(mathFunction(2, 3))")
// prints "Result: 5"
```

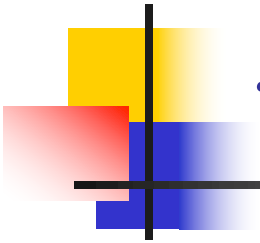
```
mathFunction = multiplyTwoInts
print("Result: \(mathFunction(2, 3))")
// prints "Result: 6"
```



Functions as parameters

Function types can be used in other functions' parameters

```
func printMathResult(_ mathFunction: (Int, Int) -> Int, _ a: Int, _ b: Int) {
    print("Result: \(mathFunction(a, b))")
}
printMathResult(addTwoInts, 3, 5)
// Prints "Result: 8"
```

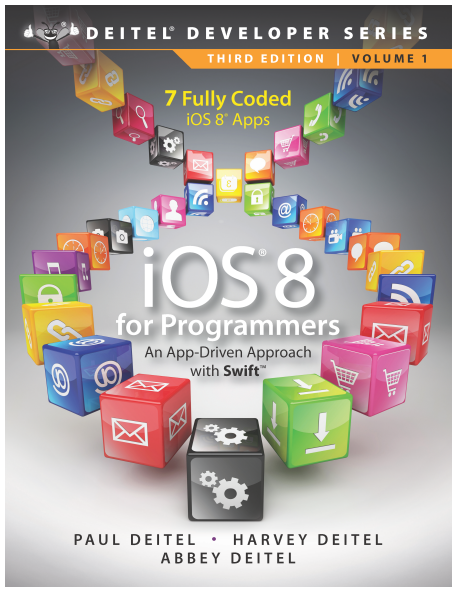


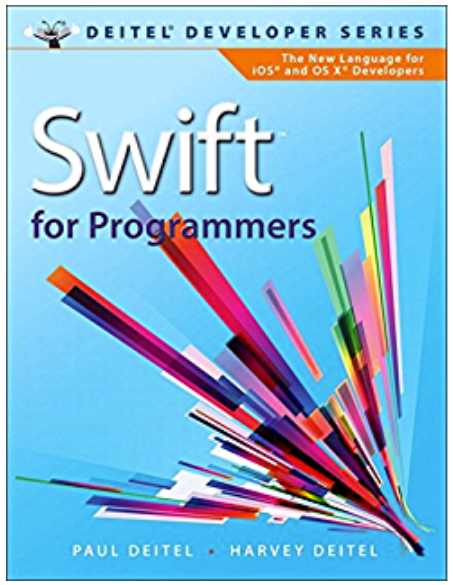
The Swift Programming Language

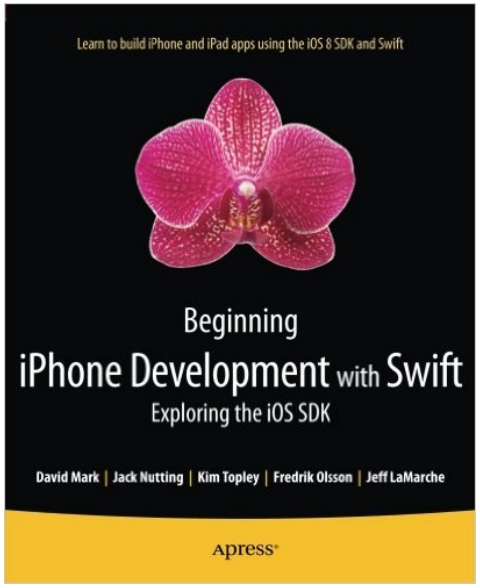
- Variables and constants must be initialized before they are used.
- No implicit conversion between numeric types
- Semicolons not required, unless used to separate multiple statements in a line.
- Parentheses around conditions not needed.
- Tuple – collection of same or different types
- Functions with multiple return values (via tuples)
- Optionals – variables which may not have value
- Type inferences based on initialized value

Resources

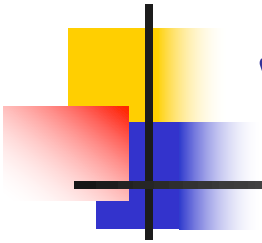
Some recommended books:







Swift portal:	https://swift.org/
Apple Developer Portal:	https://developer.apple.com/swift/resources/
Online Swift compiler:	http://www.runswiftlang.com/
Swift for Windows:	https://swiftforwindows.codeplex.com/
Swift Tutorials:	https://www.hackingwithswift.com/



Warm up Challenge

1. Write a Swift function called "division" that takes two integers and returns both the quotient and the remainder in dividing the first one by second one.

Example call of function:

division(19,3) should return (6, 1)

Warm up Challenge

1. Write a Swift function called "division" that takes two integers and returns both the quotient and the remainder in dividing the first one by second one.

Example call of function:

division(19,3) should return (6, 1)

```
func division(_ firstNumber: Int, _ secondNumber: Int) -> (Int, Int)
{
        let quotient = firstNumber/secondNumber
        let remainder = firstNumber%secondNumber

         return (quotient, remainder)
}
```

Challenge #2

2. Match the following function calls with their appropriate function definitions. If no matching definition found, write a matching function definition.

Function Calls	Matching Definition
A. sum(a: 3, b: -4)	
B. sum(7, b:2)	
C. sum(a: 3, 4)	
D. sum(5,8)	
E. sum(p1: 5, 12)	

Function Definitions

```
1. func sum(_ a: Int, _ b:Int)-> Int{ return 1}
2. func sum(a: Int, b:Int)-> Int{ return 2}
3. func sum(p1 a: Int, _ b:Int)-> Int{ return 3}
4. func sum(_ p1: Int, b:Int)-> Int{ return 4}
```

Challenge #2

2. Match the following function calls with their appropriate function definitions. If no matching definition found, write a matching function definition.

Function Calls	Matching Definition
A. sum(a: 3, b: -4)	2
B. sum(7, b:2)	4
C. sum(a: 3, 4)	5
D. sum(5,8)	1
E. sum(p1: 5, 12)	3

Function Definitions

```
1. func sum(_ a: Int, _ b:Int)-> Int{ return 1}
2. func sum(a: Int, b:Int)-> Int{ return 2}
3. func sum(p1 a: Int, _ b:Int)-> Int{ return 3}
4. func sum(_ p1: Int, b:Int)-> Int{ return 4}
5. func sum(a: Int, _ b:Int)-> Int{ return 5}
```