# Android App Links

[First app](https://www.youtube.com/watch?v=EOfCEhWq8sg),

# OiS App

Links: [First app video](https://www.youtube.com/watch?v=5b91dFhZz0g), [get started](https://developer.apple.com/library/archive/referencelibrary/GettingStarted/DevelopiOSAppsSwift/ImplementingACustomControl.html#//apple_ref/doc/uid/TP40015214-CH19-SW1), [Login](https://www.youtube.com/watch?v=a5pzlbBnfYg), [SQLite video](https://www.youtube.com/watch?v=c4wLS9py1rU), [SQLite tutorial](https://www.raywenderlich.com/385-sqlite-with-swift-tutorial-getting-started), [Why SQLite](https://www.sqlite.org/whentouse.html)

Q: The **weak keyword** indicates that the reference does not prevent the system from deallocating the referenced object. Weak references help prevent reference cycles; however, to keep the object alive and in memory you need to make sure some other part of your app has a strong reference to the object. In this case, it’s the text field’s superview. A superview maintains a **strong reference** to all of its subviews. As long as the superview remains alive and in memory, all of the subviews remain alive as well. Similarly, the view controller has a strong reference to its content view—keeping the entire view hierarchy alive and in memory.

viewDidLoad() to create and load. usually only once

viewWillAppear() just before the content view is presented onscreen(not guaranteed)

viewDidAppear() as soon as the view is presented onscreen

viewWillDisappear() just before disappear. To perform cleanup tasks (EG. committing changes or resigning the first responder status)

viewDidDisappear() just after disappear. To perform additional teardown activities

present(\_:animated:completion:) Passing true to the animated parameter animates the presentation of the image picker controller. The completion parameter refers to a completion handler. Nil if nothing to handle

Notes

* NSException => delete the connection of a button
* UITextFieldDelegate added to get user’s input
* After hitting return, textFieldShouldReturn() reassigns the first responder, then textFieldDidEndEditing() reads the input
* field.resignFirstResponder() needed to disable keyword when typing

# [Swift](https://docs.swift.org/swift-book/LanguageGuide/TheBasics.html)

## Basics

Variable

| let constantVar = 10 var variable = UInt32.max var thisstring: UInt8 = 1 print("this string is \(thisstring)") typealias AudioSample = UInt16 var maxAmplitudeFound = AudioSample.min |
| --- |

Comment

| // one-line comment /\* multiple-line comment \*/ |
| --- |

Boolean

| let orange = true if orange { // execute } var i = 1 if i { //error } if i == 1 { //execute } |
| --- |

Tuple

| let http404Error = (404, "Not Found") let (statusCode, statusMessage) = http404Error print("The status code is \(statusCode)")  // Prints "The status code is 404" let (justTheStatusCode, \_) = http404Error print("The status code is \(justTheStatusCode)")  // Prints "The status code is 404" print("The status code is \(http404Error.0)")  // Prints "The status code is 404" let http200Status = (statusCode: 200, description: "OK") print("The status code is \(http200Status.statusCode)")  // Prints "The status code is 200" |
| --- |

Optional variable with ?

| var serverResponseCode: Int? = 404 // serverResponseCode contains an actual Int value of 404 serverResponseCode = nil if let firstNumber = Int("4"), let secondNumber = Int("42"), firstNumber < secondNumber && secondNumber < 100 {  print("\(firstNumber) < \(secondNumber) < 100") } |
| --- |

Implicitly Unwrapped Optionals

| let possibleString: String? = "An optional string." let forcedString: String = possibleString! // exclamation mark |
| --- |

Do try catch

| do {  try makeASandwich()  eatASandwich() } catch SandwichError.outOfCleanDishes {  washDishes() } catch SandwichError.missingIngredients(let ingredients) {  buyGroceries(ingredients) } |
| --- |

Loop

| for index in 1...5 {  print("\(index) times 5 is \(index \* 5)") } for i in 0..<count { } for name in names[2...] { //array[2] to end } for name in names[...2] { //array[0] to array[2] } for name in names[..<2] { //array[0] to array[1] } |
| --- |

## String

| var variableString = "Horse" variableString += " and carriage"  let exclamationMark: Character = "!" variableString.append(exclamationMark) //append a char  print("length \(variableString.count)") let greeting = "Guten Tag!" greeting[greeting.startIndex] // G greeting[greeting.index(before: greeting.endIndex)] // ! greeting[greeting.index(after: greeting.startIndex)] // u  for index in greeting.indices {  print("\(greeting[index]) ", terminator: "") } // Prints "G u t e n T a g ! " |
| --- |

| //inserting var welcome = "hello" welcome.insert("!", at: welcome.endIndex) //"hello!" welcome.insert(contentsOf: " there", at: welcome.index(before: welcome.endIndex))  //"hello there!"  //removing welcome.remove(at: welcome.index(before: welcome.endIndex)) //"hello there" welcome.removeSubrange(" there") //"hello"  //substring let greeting = "Hello, world!" let index = greeting.firstIndex(of: ",") ?? greeting.endIndex  string.hasPrefix() to check the first several chars string.hasSuffix() to check the last several chars |
| --- |

## Collection

### Array

| var name = [int]() //empty int arry var threeDoubles = Array(repeating: 0.0, count: 3) var shoppingList: [String] = ["Eggs", "Milk"]  //var shoppingList = ["Eggs", "Milk"]  shoppingList = ["Eggs", "Milk", "Flour", "Baking Powder", "Chocolate Spread", "Cheese", "Butter"] //7 items shoppingList[4...6] = ["Bananas", "Apples"] //6 items now  shoppingList.insert("Maple Syrup", at: 0) //7 items let mapleSyrup = shoppingList.remove(at: 0) let apples = shoppingList.removeLast()  for item in shoppingList {  print(item) } for (index, value) in shoppingList.enumerated() {  print("Item \(index + 1): \(value)") } |
| --- |

### Set

* distinct value unordered
* a.intersection(b) a.symmetricDifference(b) a.union(b) a.subtracting(b) a.isSubset(of:b) a.isSuperset(of:b) a.isStrictSubset(of:) to determine whether a set is a subset or superset, but not equal to, a specified set a.isDisjoint(with:b) to determine whether two sets have no values in common.

| var letters = Set<Character>() letters.insert("a") letters = []  var favoriteGenres: Set<String> = ["Rock", "Classical", "Hip hop"] //var favoriteGenres: Set = ["Rock", "Classical", "Hip hop"]  favoriteGenres.insert("Jazz") let removedGenre = favoriteGenres.remove("Rock") favoriteGenres.contains("Funk")  for genre in favoriteGenres {  print("\(genre)") } for genre in favoriteGenres.sorted() {  print("\(genre)") } |
| --- |

### Dictionary

| var airports = [String: String]() airports["16"] = "sixteen" airports = [:] //empty again  airports = ["YYZ": "Toronto Pearson", "DUB": "Dublin"] airports["LHR"] = "London" //3 items let oldValue = airports.updateValue("Dublin Airport", forKey: "DUB") let removedValue = airports.removeValue(forKey: "DUB")  for (airportCode, airportName) in airports {  print("\(airportCode): \(airportName)") } for airportCode in airports.keys {  print("Airport code: \(airportCode)") } for airportName in airports.values {  print("Airport name: \(airportName)") } let airportCodes = [String](airports.keys) let airportNames = [String](airports.values) |
| --- |

## Control flow

| //For in loop for \_ in 1...5 { //i value is not important  answer \*= base } for tickMark in stride(from: 0, to: 60, by: 5) {  //to: 60(not included) (0, 5, 10, 15 ... 45, 50, 55) } for tickMark in stride(from: 0, through: 60, by: 5) {  //through: 60(included) (0, 5, 10, 15 ... 45, 50, 55) } //while loop while i < 5 { }  //repeat while loop repeat{ }while i < 5 //conditional if i < 5{ } else if i < 10 { } else { } |
| --- |
| //switch  let someCharacter: Character = "z" switch someCharacter { case "a", "A":  print("The first letter of the alphabet") case 1..<13:  print("The last letter of the alphabet") default: //otherwise  print("Some other character") } |

Function

| //Omitting Argument Labels 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)  //default func someFunction(parameterWithoutDefault: Int, parameterWithDefault: Int = 12) {  // If you omit the second argument when calling this function, then  // the value of parameterWithDefault is 12 inside the function body. } someFunction(parameterWithoutDefault: 3, parameterWithDefault: 6) someFunction(parameterWithoutDefault: 4)  //0 to more doubles func arithmeticMean(\_ numbers: Double...) -> Double {  var total: Double = 0  for number in numbers {  total += number  }  return total / Double(numbers.count) } arithmeticMean(1, 2, 3, 4, 5)  //pass the address func swapTwoInts(\_ a: inout Int, \_ b: inout Int) {  let temporaryA = a  a = b  b = temporaryA } var someInt = 3 var anotherInt = 107 swapTwoInts(&someInt, &anotherInt) |
| --- |

# Hackathon

“Available Seats” App plan:

1. Login Page(unity id + password)
   1. optional: password complexity → Security
2. Select the building Page (buttons for now → search bar if more buildings)
   1. scan the code on the top
3. Floor Plan Page(with empty seats %)
4. Reserve Page( → multi threat problem)