WELCOME

INSTRUCTOR'S INTRODUCTION:

NAME: SUJEET LOHAN

EDUCATION: MASTERS IN INFORMATION TECHNOLOGY

UNIVERSITY: INDIAN INSTITUTE OF TECHNOLOGY (IIT) ROORKEE, INDIA

EXPERIENCE : 7 YEARS IN IT INDUSTRY, 16 YEARS IN TEACHING

==========

COURSE INFO:

TITLE: MOBILE APP DEVELOPMENT

COURSE CODE: COMP2125

TERM: SUMMER 2021

Swift Programming – Key Aspects

- Swift was announced at Apple's World Wide Developer Conference(WWDC) in June 2014.
- Apple's Language of the Future for app and system programming
- **Popular Language Features** – Simpler syntax than Objective-C.
  - Designers included popular feature from other languages such as Java, C#, Ruby and others
    - **tuples**
    - **closures (lambdas)**
    - **generics**
    - **operator overloading**
    - **functions with multiple return values – possible with tuples**
    - **optionals and string interpolation etc.**
    - **Arrays**

Swift Programming – Key Aspects contd …

- Performance – Swift was designed for better performance than Objective-C. 1.5 times faster.
- Error Prevention – eliminates many programming errors, making code more robust and secure. Error prevention features include:
  - Automatic Memory Management
  - No pointers
  - Required braces around for every control statement body
  - Assignment operators that don't return values
  - Requiring initialization of all the variables and constants
  - Array bound checking and automatic checking for overflow of integers calculations and many more
- Interoperability with Objective-C

Swift Programming – Key Aspects contd ...

Swift feature	Description
Type inference	Though Swift is a strongly typed language, in many cases you do not need to specify a variable's or constant's type—Swift can *infer* the type based on the variable's or constant's initializer value.
`switch` statement enhancements	Unlike `switch` statements in other C-based languages, Swift's `switch` statement can test values of any type. Also, its cases are much more flexible than those in other languages—you can have cases for individual values, sets of values and ranges of values. You can also specify boolean criteria that must be true for a match to occur.
Closures	Swift supports functional-programming techniques via closures (anonymous functions that some languages call *lambdas*). Closures can be manipulated as data—they can be assigned to variables, passed to functions as arguments and returned from functions. Several of the Swift Standard Library's global functions receive closures as arguments—for example, there's a version of the `sort` function that receives a closure for comparing two objects to determine their sort order.

Swift Programming – Key Aspects contd ...

Tuples	Swift provides *tuples*—collections of values that can be of the same or different types. The language provides syntax for *composing* (creating) and *decomposing* (extracting values from) a tuple.
Optionals	*Optionals* enable you to define variables and constants that might not have a value. The language provides mechanisms for determining whether an optional has a value and, if so, obtaining that value. Optionals work for any Swift type, whereas the corresponding concept in Objective-C—a pointer that points to an object or is `nil`—works only for reference types.
`Dictionary` type	Swift's `Dictionary` type provides built-in support for manipulating data in *key–value pairs*.
`Array`, `String` and `Dictionary` value types	Swift types `Array`, `String` and `Dictionary` are *value* types (not reference types as you might expect) that are implemented as `structs`. Objects of value types are *copied* when you assign them to variables or constants, pass them to functions or return them from functions. The Swift compiler optimizes value-type copy operations, performing them only when necessary.

Swift Programming – Key Aspects contd …

Array bounds checking	A *runtime* error occurs if you access an element outside an Array's bounds.
Class-like struct and enum value types	Swift's struct and enum types have many class-like features, making them more robust than their Objective-C counterparts. Objects of struct and enum types are *value* types.
Functions with multiple return values (via tuples)	Functions can return multiple values of possibly different types as tuples.
Generics	Rather than writing separate code to perform identical tasks on different types (e.g., summing an Array of integers vs. summing an Array of floating-point values), *generics* enable you to write the code once and use placeholders to represent the type(s) of data to manipulate. The compiler substitutes actual types for the placeholders. In a generic function call, the compiler determines the actual types based on the arguments specified in the code that calls the function. For a generic type, the compiler uses the actual type you specify when declaring an object of that generic type. Swift's Array and Dictionary types are generic types, and many of its global functions are generic functions. When you use generic functions and types, compile-time type checking is performed to ensure that you use the functions and types correctly. For example, creating an Array of integers, then attempting to place a String into that Array is a compilation error.

Swift Programming – Key Aspects contd ...

Operator overloading	You can define functions that overload existing operators to work with new types, and you can also define *entirely new operators*—which you cannot do in C# or C++, for example.
Overflow checking in integer calculations	*By default*, all integer calculations check for *arithmetic overflow* and result in a runtime error if overflow occurs.
`String` interpolation	`String` interpolation enables you to build `Strings` by *inserting* variable, constant and expression values into *placeholders* directly in `String` literals.
Nested types	You can define types *nested* in other type definitions—this is commonly used to define `enums` or utility classes and `structs` that are *hidden* in the scope of another type.
Nested functions	You can nest function definitions in other function definitions—such a nested function is callable in the scope of its enclosing function and can be returned from that function for use in other scopes.

Swift Programming – Error prevention features

Some Swift features that eliminate common programming errors

- Curly braces ({}) are required around *every* control statement's body. For all control statements this helps ensure that you do not accidentally forget the braces around multi-statement bodies.
- Unlike Objective-C, C and C++, Swift does not include pointers.
- The assignment operator (=) does not return a value. A compilation error occurs if = is used in a condition rather than the equal-to operator (==), thus preventing the common error of typing a single = when you intend to use the equal-to operator (==).
- Semicolons are optional unless you need to separate multiple statements on the same line.
- Parentheses around conditions in control statements are optional, making the code a bit easier to read.
- Variables and constants *must* be initialized before they're used—either in their definitions or via initializer methods in type definitions.

Swift Programming – Key Aspects contd ...

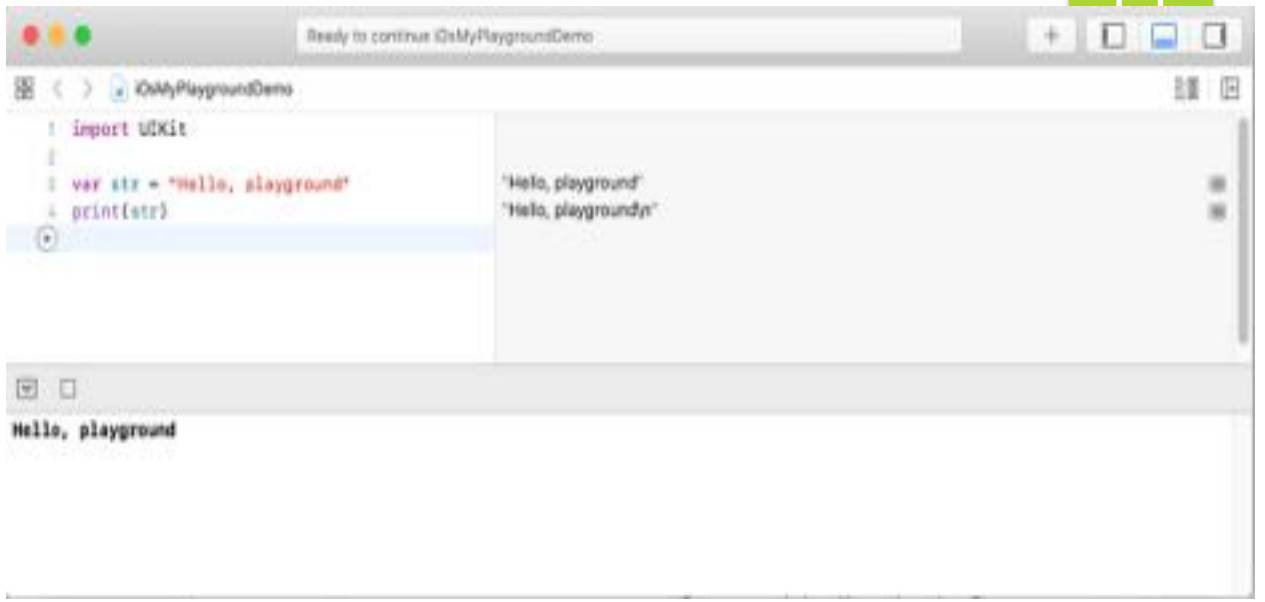
- Integer calculations are *checked for overflow by default*—a runtime error occurs if a calculation results in overflow.
- Swift *does not allow implicit conversions* between numeric types.
- `Array` indices (subscripts) are *bounds checked* at execution time—a runtime error occurs if you access an element outside an `Array`'s bounds.
- Automatic memory management *eliminates most memory leaks*—it's still possible to maintain references to objects that are no longer used and thus prevent their memory from being reclaimed by the runtime. Swift also has *weak references* for cases in which circular references between objects would prevent those objects' memory from being reclaimed.

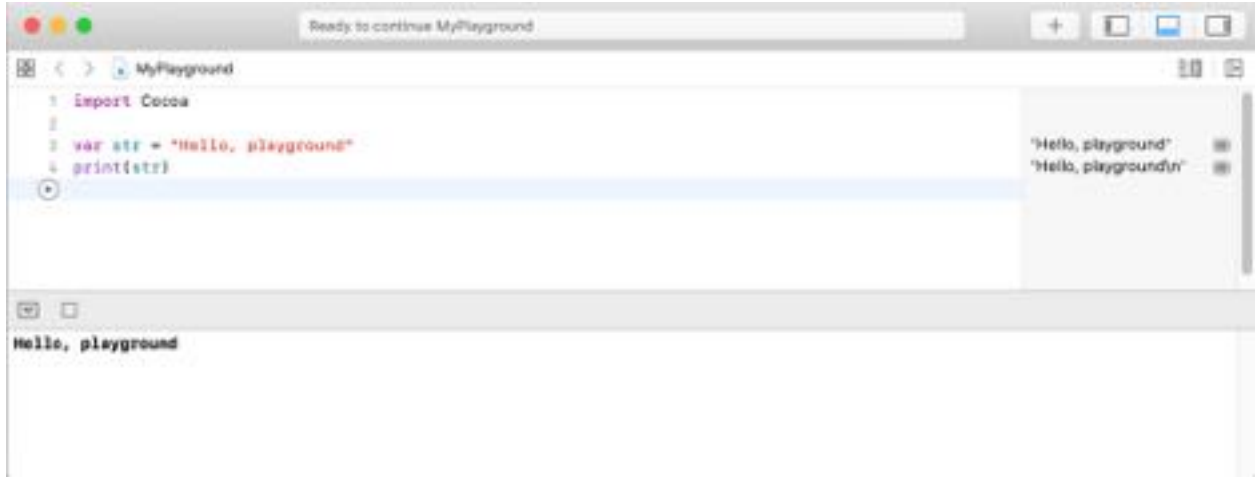
Swift Programming – Key Aspects contd …

- **Playgrounds:**
  - It is an Xcode window on which you can enter swift code and execute as you type it.
  - A playground does not require that you compile and run a complete project.
  - This allows you to see and hear your code's results as you write it and quickly fix the errors
  - Ideal for testing and experimenting with the Swift language in the lightweight environment
  - Extension of a playground file is **.playground** (e.g. welcome.playground)

Swift Programming – Key Aspects contd …

- **Playgrounds:**





IDE for Swift Programming

- Swift version 4.0/5.0
- Xcode – version 11.0 (11.4.x)

  - You can down load it from https://developer.apple.com/xcode -- latest version is 11.4
  - Tightly integrated with the Cocoa and Cocoa Touch frameworks, Xcode is an incredibly productive environment for building apps for Mac, iPhone, iPad, Apple Watch, and Apple TV.
- Resources:
  - The swift programming language - available in iBooks store and at: https:/developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language

IDE for Swift Programming contd...

- Cocoa Framework
- Cocoa Touch Framework

These are reusable components, built-in libraries.

- Swift, like Objective-C, can use macOS's Cocoa frameworks and iOS's Cocoa Touch frameworks.
- These powerful libraries of prebuilt components help you create apps that meet Apple's requirements for the look-and-feel of iOS and macOS apps.
- The frameworks are written mainly in Objective-C (some are written in C);

Frameworks

Three key frameworks for macOS and iOS development are **Foundation**, **AppKit** and **UIKit.**

- **The Foundation framework** —in both Cocoa and Cocoa Touch—includes class NSObject for defining object behavior.
- Foundation also has classes for basic types, storing data, working with text and strings, filesystem access, calculating differences in dates and times, inter-app notifications and much more.

Cocoa's AppKit framework is for developing the UIs of macOS apps.

- AppKit provides controls (such as windows, menus, buttons, panels, text fields, dialogs), event-handling capabilities, gesture support and more.
- It also supports content sharing between services (e.g., e-mail), iCloud integration, printing, accessibility (for users with disabilities), push notifications, graphics and more.

Frameworks

UIKit Framework

- Cocoa Touch's UIKit framework is similar to AppKit, but optimized for developing iOS app UIs for mobile devices.
- UIKit includes multi-touch interface controls that are appropriate for mobile apps, event handling for motion-based events, event handling for sensors (e.g., proximity, motion, accelerometer, ambient light, gyroscope) and more

Other Cocoa touch Frameworks

Framework	Description
AddressBookUI	Display contact information from the user's Address Book.
EventKitUI	Edit, create and display calendar events from within your apps.
GameKit	Voice, Bluetooth networking and other capabilities that can be used in games or other apps.
MapKit	Add maps and satellite images to location-based apps. Annotate maps, identify areas on a map using overlays and more.
MessageUI	Create e-mail messages from within an app. Create and send SMS messages from within an app.
NotificationCenter	Display information from your app in Notification Center and allow users to perform brief tasks for your app (such as responding to a message).
PhotosUI	Incorporate iOS photo and video editing features into your own apps.
Twitter	Add tweeting capabilities to any app.
UIKit	Classes for creating and managing a user interface, including event handling, drawing, windows, views and multi-touch interface controls. Introduced in Chapter 2's **Welcome** app and used throughout the book.
iAd	In-app advertising framework used to place full-screen or banner advertisements within an app for monetization.

References/Resources:

- https://developer.apple.com/swift/
- https://docs.swift.org/swift-book
- **Swift for Programming by Deitel and Deitel**
- **Swift Programming 2nd Edition by Matthew Mathias and John Gallagher**
- **IOS Programming 6th Edition by Christian Keur and Aaron Hillegass**
- https://www.tutorialspoint.com/swift/index.htm
- www.udemy.com
- **Edureka.co**
- **Code Repositories on GitHub**
- **Resources on internet**