Swift Complete Course

By:

PASS Education System

What is Swift?

History

Swift was developed by Apple Inc. and the open-source community as a replacement for Apple's earlier programming language Objective-C, which had been largely unchanged since the early 1980s and lacked modern language features1. Swift was announced in 2014 at Apple's Worldwide Developer Conference (WWDC) and has quickly become one of the fastest growing languages in history2. Swift works with Apple's Cocoa and Cocoa Touch frameworks, and can interoperate with existing Objective-C code1. Swift is designed to be safe, fast, expressive, and modern, with features such as closures, generics, protocols, optionals, and memory management123. Swift is also an open language that lets everyone contribute to its development2. Swift is currently the preferred programming language for Apple platforms, and is also available for other operating systems such as Linux, Windows, and Android.

What is Swift (FEATURES) ?

Swift has several features that make it a powerful and intuitive programming language for all Apple platforms. Some of these features are:

- ***Closures unified with function pointers***
- ***Tuples and multiple return values***
- ***Generics that are powerful and simple to use***
- ***Fast and concise iteration over a range or collection***
- ***Structs that support methods, extensions, and protocols***
- ***Functional programming patterns, e.g., map and filter***

- ***Powerful error handling built-in***
- ***Advanced control flow with do, guard, defer, and repeat keywords***
- ***Memory safety and management using Automatic Reference Counting (ARC)***
- ***Protocol extensions that make writing generic code even easier***
- ***Flexible enumerations that support pattern matching and can have payloads***
- ***Package manager that is a cross-platform tool for building, running, testing, and packaging Swift libraries and executables.***
- ***Debugging using the LLDB debugger, which provides a REPL and debugger to enable integrated debugging, consistent formatting, failure recovery, and expression evaluation.***
- ***Performance that is fast and efficient, comparable to C-based languages.***

Applications of HTML?

Swift is mainly used for developing mobile and desktop applications for Apple's devices, such as iOS, macOS, watchOS, and tvOS123. It was created to make writing, maintaining, and correcting programs easier for developers1. Some of the applications that are written in Swift are:

LinkedIn: A professional networking app that connects millions of people around the world

Lyft: A ride-sharing app that offers transportation services in hundreds of cities

WordPress: A popular blogging platform that powers over 40% of the web

VSCO: A photography app that lets users edit and share photos with creative filters and tools

Clear Sky Guide: An astronomy app that helps users identify stars, planets, and constellations

Swift is also a general-purpose programming language that can be used for other domains, such as systems programming, cloud services, machine learning, and scripting5. Swift is open source and has a growing community of developers who contribute to its development.

Basics of Swift? / Requirements for Swift / How to Start Swift?

To start programming with Swift, you'll need to set up your development environment and learn the basics of the Swift programming language. Here are the steps to get started:

Install Xcode: Xcode is the integrated development environment (IDE) for Swift and iOS/macOS development. You can download it for free from the Mac App Store.

Learn the Basics of Swift: Swift is a modern, powerful, and user-friendly programming language. Familiarize yourself with the language syntax, data types, control flow statements, functions, and other fundamental concepts. There are many resources available online, including Apple's official Swift documentation, Swift Playgrounds app, and tutorials on websites like Ray Wenderlich and Hacking with Swift.

Set Up a New Project: Launch Xcode and create a new Swift project. Xcode provides project templates for various types of applications, including iOS, macOS, watchOS, and tvOS. Choose the appropriate template based on your desired platform and project type.

Explore the Xcode Interface: Familiarize yourself with the Xcode interface, including the editor area, navigator area, utility area, and debug area. Learn how to build and run your project, set breakpoints, and navigate through your code.

Write Swift Code: Start writing Swift code in the appropriate files (e.g., Swift source files with a .swift extension). Xcode provides code completion, syntax highlighting, and various productivity features to help you write code efficiently.

Practice with Simple Examples: Begin with small, manageable coding examples to solidify your understanding of Swift concepts. Experiment with variables, constants, functions, conditionals, loops, arrays, dictionaries, and other Swift language features.

Utilize Online Resources: Take advantage of online resources like tutorials, blogs, forums, and Swift programming communities. These can provide additional guidance, code samples, and answers to your questions as you progress.

Build Simple Apps: As you gain confidence in Swift, try building simple apps or adding functionality to existing projects. Start with basic user interfaces, handle user interactions, and explore iOS frameworks and APIs.

Iterate and Expand: Swift is continuously evolving, so keep learning and stay updated with the latest Swift versions and features. Attend local developer meetups or join online communities to connect with other Swift developers and share knowledge.

Basics Operators for Swift?

Here are the basic operators in the Swift programming language:

Arithmetic Operators:

Addition: +

Subtraction: -

Multiplication: *

Division: /

Remainder: %

Assignment Operators:

Assignment: =

Addition assignment: +=

Subtraction assignment: -=

Multiplication assignment: *=

Division assignment: /=

Remainder assignment: %=

Comparison Operators:

Equal to: ==

Not equal to: !=

Greater than: >

Less than: <

Greater than or equal to: >=

Less than or equal to: <=

Logical Operators:

Logical NOT: !

Logical AND: &&

Logical OR: ||

Compound Operators:

Compound addition: +=

Compound subtraction: -=

Compound multiplication: *=

Compound division: /=

Compound remainder: %=

Range Operators:

Closed range operator: ...

Half-open range operator: ..<

Ternary Conditional Operator:

(condition) ? valueIfTrue : valueIfFalse

Nil-Coalescing Operator:

Nil-coalescing operator: a ?? b

Type Casting Operators:

Type check operator: is

Type cast operator: as

Basics Variables for Swift?

In Swift, you can declare and use various types of variables. Here are the basic variable types in Swift:

Integer Types:

Int: Represents whole numbers, both positive and negative.

UInt: Represents unsigned (non-negative) whole numbers.

Floating-Point Types:

Float: Represents single-precision floating-point numbers.

Double: Represents double-precision floating-point numbers.

Boolean Type:

Bool: Represents boolean values, either true or false.

Character Type:

Character: Represents a single character.

String Type:

String: Represents a sequence of characters.

Array Type:

Array: Represents an ordered collection of values of the same type.

Example: var numbers: [Int] = [1, 2, 3, 4]

Dictionary Type:

Dictionary: Represents a collection of key-value pairs.

Example: var scores: [String: Int] = ["Alice": 95, "Bob": 80, "Charlie": 75]

Tuple Type:

Tuple: Represents a grouping of values of different types.

Example: var person: (String, Int) = ("Alice", 25)

Optional Type:

Optional: Represents a variable that can either hold a value of a specific type or be nil.

Example: var optionalValue: Int? = 5

Implicitly Unwrapped Optional Type:

ImplicitlyUnwrappedOptional: Represents an optional variable that is assumed to always have a value and doesn't require unwrapping.

Example: var unwrappedValue: String! = "Hello"

These are the basic variable types in Swift. Variables can be declared using the var keyword for mutable values (can be changed), or the let keyword for immutable values (constants). Remember to provide an appropriate type annotation when declaring variables, or Swift can infer the type based on the assigned value.

Basics Functions for Swift?

Swift provides a variety of built-in functions that you can use in your code. Here are some commonly used functions in the Swift programming language:

Print Functions:

print(_:separator:terminator:): Prints one or more values to the console.

debugPrint(_:separator:terminator:): Prints a debug representation of a value to the console.

String Functions:

count: Returns the number of characters in a string.

isEmpty: Checks if a string is empty.

hasPrefix(_:)/hasSuffix(_:): Checks if a string has a specified prefix/suffix.

lowercased()/uppercased(): Converts a string to lowercase/uppercase.

split(separator:)/components(separatedBy:): Splits a string into an array based on a separator.

Array Functions:

count: Returns the number of elements in an array.

isEmpty: Checks if an array is empty.

append(_:): Adds an element to the end of an array.

insert(_:at:): Inserts an element at a specific index in an array.

remove(at:): Removes an element at a specific index from an array.

first/last: Returns the first/last element of an array.

contains(_:): Checks if an array contains a specific element.

map(_:): Transforms each element of an array using a closure.

filter(_:): Creates a new array with elements that satisfy a condition.

Dictionary Functions:

count: Returns the number of key-value pairs in a dictionary.

isEmpty: Checks if a dictionary is empty.

keys/values: Returns an array of all keys/values in a dictionary.

containsKey(_:): Checks if a dictionary contains a specific key.

updateValue(_:forKey:): Updates the value for a specific key in a dictionary.

removeValue(forKey:): Removes a key-value pair for a specific key from a dictionary.

Math Functions:

abs(_:): Returns the absolute value of a number.

max(_:_:): Returns the maximum value between two numbers.

min(_:_:): Returns the minimum value between two numbers.

sqrt(_:): Returns the square root of a number.

random(in:): Generates a random number within a specified range.

Control Flow Functions:

if/else: Executes code based on a condition.

switch/case: Matches a value against multiple patterns.

for-in: Iterates over a sequence (array, range, string, etc.).

while/repeat-while: Executes code repeatedly while a condition is true.

These are just a few examples of the built-in functions available in Swift. Additionally, Swift allows you to define your own custom functions to encapsulate reusable code.

How to Run Swift Code?

To run Swift code, you can follow these steps:

Install Xcode: Xcode is the integrated development environment (IDE) for Swift and iOS/macOS development. You can download it for free from the Mac App Store. Make sure you have Xcode installed on your Mac.

Open Xcode: Launch Xcode by clicking on its icon in the Applications folder or by searching for it in Spotlight.

Create a New Project or Playground: In Xcode, you have the option to either create a new project or use a playground to run Swift code. Playgrounds are a great way to experiment with Swift code snippets and see the results in real-time. To create a new playground, go to "File" -> "New" -> "Playground" and follow the prompts. If you want to create a full project, select the appropriate project template based on your desired platform (iOS, macOS, etc.) and follow the setup steps.

Write Swift Code: In the Xcode editor, you can start writing your Swift code. For playgrounds, you can write code directly in the code editor. For projects, you'll need to create Swift source files (with a .swift extension) and write your code there.

Build and Run: To run your Swift code, you can click the "Play" button (a triangle icon) in the Xcode toolbar. Xcode will build your project and run the code. For playgrounds, you'll see the output and results in the right-hand side panel as you write and modify your code.

View the Output: After running your Swift code, you can view the output in the console area at the bottom of the Xcode window. Any print statements or debug output will be displayed here.

Alternatively, you can also run Swift code directly in the Terminal without Xcode. Open the Terminal app on your Mac, navigate to the directory where your Swift code file is located, and use the swift command followed by the name of the Swift file to run it. For example:

```
swift MyCode.swift
```

Replace "MyCode.swift" with the actual name of your Swift file.

These steps should help you run Swift code either in Xcode or in the Terminal.

Swift Projects :

Here are some basic project ideas for beginners to practice and enhance their Swift programming skills:

Calculator: Build a simple calculator app that can perform basic arithmetic operations like addition, subtraction, multiplication, and division.

Tip Calculator: Create an app that calculates the tip amount based on the bill total and allows users to adjust the tip percentage.

To-Do List: Develop a to-do list app where users can add, delete, and mark tasks as complete.

Weather App: Build a weather app that fetches and displays weather data for a specific location using an API. Users can enter a city or use GPS to get the current weather information.

Random Quotes Generator: Create an app that generates and displays random quotes when a button is pressed. You can store the quotes in an array and display them randomly.

Guessing Game: Develop a number guessing game where the app generates a random number, and the user tries to guess it. Provide feedback on whether the guess is too high or too low until the user guesses correctly.

Unit Converter: Build a unit converter app that can convert measurements between different units, such as length, weight, temperature, or currency.

Flashcards: Create a flashcard app that allows users to create decks of flashcards for studying. Users can flip the cards to view questions and answers.

Photo Gallery: Develop a simple photo gallery app that displays a collection of images in a grid view and allows users to select and view individual images in full screen.

BMI Calculator: Build a body mass index (BMI) calculator app that takes user input for height and weight and calculates the BMI value along with a corresponding classification (e.g., underweight, normal, overweight).

These project ideas cover a range of concepts and provide opportunities to practice Swift fundamentals, user interface design, data handling, and API integration. Start with a project that interests you and gradually increase the complexity as you gain more experience. Remember to break down the project into smaller tasks and tackle them one at a time. Happy coding!

- Weather App

- Tip Calculator

- Make Your Own for Practice

Project: Weather App

Here's a basic example of Swift code for a weather app. This code demonstrates fetching weather data from the OpenWeatherMap API and updating the UI with the retrieved information.

```
import UIKit

struct WeatherData: Codable {
    let name: String
    let main: Main
    let weather: [Weather]
}

struct Main: Codable {
    let temp: Double
}

struct Weather: Codable {
    let description: String
    let icon: String
}

class WeatherViewController: UIViewController {

    @IBOutlet weak var cityLabel: UILabel!
    @IBOutlet weak var temperatureLabel: UILabel!
    @IBOutlet weak var descriptionLabel: UILabel!
    @IBOutlet weak var iconImageView: UIImageView!

    let apiKey = "YOUR_API_KEY"
    let baseURL = "https://api.openweathermap.org/data/2.5/weather"
```

```
    override func viewDidLoad() {
        super.viewDidLoad()

        // Set up initial UI state
        cityLabel.text = ""
        temperatureLabel.text = ""
        descriptionLabel.text = ""
        iconImageView.image = nil

        // Call the API to fetch weather data
        fetchWeatherData()
    }

    func fetchWeatherData() {
        guard let url = URL(string: "\(baseURL)?q=London&appid=\(apiKey)") else {
            return
        }

        let task = URLSession.shared.dataTask(with: url) { (data, response, error) in
            if let error = error {
                print("Error: \(error.localizedDescription)")
                return
            }

            guard let data = data else {
                print("No data received.")
                return
            }
```

```
            do {
                let decoder = JSONDecoder()
                let weatherData = try decoder.decode(WeatherData.self, from:
data)
                self.updateUI(with: weatherData)
            } catch {
                print("Error decoding JSON: \(error.localizedDescription)")
            }
        }

        task.resume()
    }

    func updateUI(with weatherData: WeatherData) {
        DispatchQueue.main.async {
            self.cityLabel.text = weatherData.name
            self.temperatureLabel.text = "\(Int(weatherData.main.temp -
273.15))°C"
            self.descriptionLabel.text =
weatherData.weather.first?.description

            if let iconCode = weatherData.weather.first?.icon {
                self.loadIcon(iconCode: iconCode)
            }
        }
    }

    func loadIcon(iconCode: String) {
        guard let url = URL(string:
"https://openweathermap.org/img/w/\(iconCode).png") else {
```

```
            return
        }

        let task = URLSession.shared.dataTask(with: url) { (data, response, error) in
            if let error = error {
                print("Error: \(error.localizedDescription)")
                return
            }

            guard let data = data, let image = UIImage(data: data) else {
                print("Failed to load icon image.")
                return
            }

            DispatchQueue.main.async {
                self.iconImageView.image = image
            }
        }

        task.resume()
    }
}
```

his code assumes you have a view controller with outlets (cityLabel, temperatureLabel, descriptionLabel, and iconImageView) connected to the appropriate UI elements in your storyboard.

Make sure to replace "YOUR_API_KEY" with your actual API key from OpenWeatherMap. Also, note that this example fetches weather data for London. You can modify the URL in fetchWeatherData() to retrieve weather data for a different location by providing the appropriate query parameter (e.g., q=New%20York for New York).

Remember to handle errors, add appropriate error handling, and customize the UI to suit your needs.

Project: Tip Calculator

Here's an example of Swift code for a basic tip calculator app:

```
import UIKit

class TipCalculatorViewController: UIViewController {

    @IBOutlet weak var billAmountTextField: UITextField!
    @IBOutlet weak var tipPercentageSlider: UISlider!
    @IBOutlet weak var tipPercentageLabel: UILabel!
    @IBOutlet weak var tipAmountLabel: UILabel!
    @IBOutlet weak var totalAmountLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Set up initial UI state
        tipPercentageSlider.value = 15
        updateTipPercentageLabel()
        updateTipAndTotalAmount()

        // Add target to update UI when bill amount changes
        billAmountTextField.addTarget(self, action:
#selector(billAmountDidChange), for: .editingChanged)
    }
```

```
    @IBAction func tipPercentageDidChange(_ sender: UISlider) {
        updateTipPercentageLabel()
        updateTipAndTotalAmount()
    }

    @objc func billAmountDidChange() {
        updateTipAndTotalAmount()
    }

    func updateTipPercentageLabel() {
        let tipPercentage = Int(tipPercentageSlider.value)
        tipPercentageLabel.text = "\(tipPercentage)%"
    }

    func updateTipAndTotalAmount() {
        guard let billAmountText = billAmountTextField.text,
              let billAmount = Double(billAmountText) else {
            tipAmountLabel.text = "$0.00"
            totalAmountLabel.text = "$0.00"
            return
        }

        let tipPercentage = Double(tipPercentageSlider.value)
        let tipAmount = billAmount * tipPercentage / 100
        let totalAmount = billAmount + tipAmount

        tipAmountLabel.text = String(format: "$%.2f", tipAmount)
        totalAmountLabel.text = String(format: "$%.2f", totalAmount)
    }
```

```
}
```

This code assumes you have a view controller with outlets (billAmountTextField, tipPercentageSlider, tipPercentageLabel, tipAmountLabel, and totalAmountLabel) connected to the appropriate UI elements in your storyboard.

The updateTipAndTotalAmount() function calculates the tip amount and total amount based on the bill amount and the selected tip percentage. It updates the corresponding labels accordingly.

The tipPercentageDidChange(_:) function is called when the user adjusts the tip percentage using the slider, and it updates the UI accordingly.

The billAmountDidChange() function is called whenever the text in the bill amount text field changes, and it triggers the update of the tip and total amounts.

You can customize the UI and add error handling as per your requirements.

Hope this helps you create a simple tip calculator app in Swift!

```
Best Of Luck
```

```
Don't Stop Until You Have Done Completely and Truly.
```