Swift Programming Workbook

Arun Patwardhan

Copyright

About this Guide

This guide is meant to be used as a part of the iOS & iPadOS App Development Course. The guide is designed to be an independently used material too if needed.

202327611082322384

Disclaimer

The information Is provided “As Is”, without warranties of any kind, express or implied, including but not limited to the warranties Of merchantability, fitness for a particular purpose and non-infringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the information or the use or other dealings in the information.

20232761108232384

Terms and conditions

Amaranthine's training course materials, including the book, are Amaranthine's confidential and proprietary information.
Such materials may not be copied, reproduced, distributed, disclosed or published by Customer or utilised for any purpose other than as required during a training course. Customer may not alter or remove any proprietary notices or legends contained in or on any of the training course materials. Violation of this policy may result in immediate dismissal from a training course, and/or barred from future training courses, in addition to any other remedies Amaranthine may have at law or in equity.

About the author

I am an Apple Certified Master Trainer. I have been doing trainings on Apple since the past 8 years. My current areas of research include App Development for Apple's Operating Systems. Apart from training. I am also involved in course design & content development.

www.arunpatwardhan.com
http://www.linkedin.com/in/arunpatwardhan
www.amaranthine.in

202327611082322384

Table of Contents

20232761108232384

1
Chapter 1: Basic Syntax

In this chapter the reader will get familiar with the basic concepts of Swift.

Note
You will be performing all the exercises in separate playground pages. Please follow the instructions carefully. You are free to try all this code in a single playground too.

202327611082322384

Exercise 1.1: Getting familiar with variables

Prerequisites

Basic programming concepts

Scope of Work

Get familiar with variables, constants, optionals, and the built in types in Swift

Tools Required

Xcode 14.x

Playground

Outcome of the exercise

You should be comfortable with variables, constants, and optionals.

Steps

1. Open Xcode.
2. Select "*Create new playground*" or click on *File > New > Playground*.
3. The first step is to create a playground page for table of contents. You will be doing this one time only. You will not have to do this for all the remaining exercises in this chapter. To create a table of contents write down the code below.

```
/*: Introduction to Swift

# Table of Contents

 * [Variables](Variables)

*/
```

4. Now click on *File > New > Playground Page*. This is going to be the first page which holds the same content as the playground file itself.
5. Create another Playground page. *File > New > Playground Page*. Going forward you will be creating playground pages for each exercise. You can view the steps here.
6. Type the following code at the start of the screen[1].

```
//:# Variables, Types & Optionals
//: - Date: <#day#>   <#month#>   <#year#>

/*:

- note: For more information about how these markups were written please visit:
[Adding formatted Text to Swift](https://arunpatwardhan.com/2017/11/09/adding-
formatted-text-to-swift-in-xcode/)
 */
import Foundation
```

[1] You may see slightly different colours on your Xcode screen. That is okay. It depends on the Font style for Xcode.

7. Next we will look at how to declare variables. The code below show how variables can be declared with implicit type inference and explicit type declaration.

```
//:#### Declaring Variables and assigning values
//: - Note: The variables declared have their type implicitly determined.
var name    = "Arun"
var age     = 33; //var count = 0
print("Hello \(name)")

var symbol = "$"

//:#### Explicity type Declaration
var priceOfOil  : Float = 33.4
let pi          : Float = 3.14159
```

To view the comments content in a formatted manner follow the steps in Appendix A: Show rendered markup.

8. The next bit of code focuses on optionals: How to declare them, give them values, extract values.

```
//:#### Optionals
/*:
 Optionals indicate the potential absence of a value.
 - Important: Optionals indicate potential absence of a value. It does NOT mean
that the value is '0'.
 */
var screenCount : Int?      //this is an optional
var count       : Int = 0   //this is not an optional

screenCount = 0
//: - Important: Always check to make sure that the optional is not nil before
unwrapping its value.
if nil != screenCount
{
    print("\(screenCount!)")
}
screenCount = 33
```

9. Impicit optionals are like optionals except they are guaranteed to have a value later on. The example below is straightforward.

```
//:#### Implicit Optionals
/*:
 Similar to optionals with a couple of differences. A: Implicit optionals are to be used when the variable is guaranteed to have a value after a certain time.
 B: Implicit Optionals need not be unwrapped.
 - Important: Attempting to access the value of a nil Implicit Optional will result in a crash.
 */
var counter : Int!

counter = 10

print("\(counter!)")
```

Exercise 1.2: Explore strings

Prerequisites

Exercise 1.1

Scope of Work

Get familiar with the different string operations.

Tools Required

Xcode

Playgrounds

Outcome of the exercise

The reader should be familiar with strings

Warning

This is a warning.

Steps

1. Create another Playground page. You can view the steps here.
2. Let us start off by declaring a string literal.

```
//:#### String Literals
var text        = "This is the second day of the Swift development course."
```

3. Declare a multiline literal as shown below.

```
//:#### Multiline Strings
let paragraph   = """
This is a paragraph.
Here i can write strings on mulitple lines.
...
"""
```

4. Now let us iterate through the string and print out all the characters.

```
//:#### Characters
for sample in paragraph {
    print(sample)
}

let charArray           : [Character]   = ["S","W","I","F","T"]

let str                 : String        = String(charArray)

print(str)
```

5. Finally we wil look at string interpolation.

```
//: String Interpolation
var age                             = 34

var weight                          = 80.1

var firstName                       = "John"

var lastName                        = "Smith"

var description     : String        = "Name: \(firstName) \(lastName), Age: \(age), Weight: \
(weight)"

var newStr                          = "\(654)"
print(newStr)
```

Exercise 1.3: Explore functions with argument labels

Prerequisites

Exercise 1.1

Scope of Work

Explore the sytax for functions

Tools Required

Playgrounds

Outcome of the exercise

You should be comfortable with the different parts of a function.

Steps

1. Create another Playground page. You can view the steps here.
2. Let us start off by declaring a function.

```
func swapThe()
{

}
```

3. Next we will add the arguments to the function. The arguments will have both an external name and an internal name.

```
func swapThe(Number num1 : inout Int, withTheNumber num2 : inout Int)
{

}
```

4. Finally we wil implement the body of the function.

```
func swapThe(Number num1 : inout Int, withTheNumber num2 : inout Int)
{
    let temp = num1
    num1 = num2
    num2 = temp
}
```

5. Now all that is left to do is to call the function.

```
var a = 2
var b = 3
swapThe(Number: &a, withTheNumber: &b)
```

Exercise 1.4: Create structs and their objects

Prerequisites

Exercise 1.1

Scope of Work

Explore the syntax for structs

Tools Required

Playgrounds

Outcome of the exercise

You should be comfortable with the different parts of a struct.

Steps

1. Create another Playground page. You can view the steps here.
2. Implement the struct as shown below.

```
struct Address {
    var streetName      : String    = ""
    var buildingNumber  : Int       = 0
    var city            : String    = ""
    var zipCode         : String    = ""
}

struct Person {
    var name            : String    = ""
    var dateOfBirth     : Date?     = nil
    var email           : String    = ""
    var homeAddress     : Address?  = nil
}
```

3. Create objects of the struct

```
var residence : Address = Address(streetName: "10th Avenue",
                                  buildingNumber: 10,
                                  city: "Los Angeles",
                                  zipCode: "90089")

var john : Person = Person(name: "John",
                           dateOfBirth: Date(timeIntervalSince1970: 8787687324),
                           email: "john@mail.com",
                           homeAddress: residence)
```

4. Now use the objects to access and manipulate data.

Exercise 1.5: Create classes and their objects

Prerequisites

Exercise 1.1

Scope of Work

Explore the syntax for classes

Tools Required

Playgrounds

Outcome of the exercise

You should be comfortable with the different parts of a struct.

Steps

1. Create another Playground page. You can view the steps here.
2. Implement a class called address as shown below.

```
//:#### Classes
class Address {
    var buildingNumber   : Int?
    var streetName       : String?
    var city             : String?
    var state            : String?
    var zipCode          : Int?

    init() {
        buildingNumber = 0
        streetName = ""
        city = ""
        state = ""
        zipCode = 0
    }
}
```

3. Implement another class called Person as shown below.

```
class Person {
    var name            : String?
    var age             : Int?
    var personsAddress  : Address?
    var dateOfBirthday  : String

    init() {
        name            = ""
        age             = 0
        personsAddress  = Address()
        dateOfBirthday  = ""
    }

    func setAge(newAge : Int) {
        if newAge > 0 && newAge < 100 {
            age = newAge
        }
    }
}
```

4. Feel free to create an instance of this class. Play around with the properties of the class. The code below shows some of the things that you could do.

```
var jack : Person                           = Person()
jack.age                                    = 1000
jack.setAge(newAge: 32)
jack.name                                   = "Jack"
jack.dateOfBirthday                         = "1st April 1986"

if nil != jack.personsAddress {
    jack.personsAddress!.buildingNumber     = 33
}
jack.personsAddress = nil
jack.personsAddress?.buildingNumber         = 33
jack.personsAddress?.state                  = "CA"
jack.personsAddress?.city                   = "Los Angeles"
jack.personsAddress?.zipCode                = 90089

if nil != jack.name {
    print("NAME: \(jack.name!)")
}
```

5. Next we will look at inheritance. Implement the base class as shown below.

```
class Car {
    static var isTechProduct    : Bool      = true
    var paxCount                : Int       = 0
    var maxSpeed                : Float     = 0.0
    var average                 : Float     = 0.0
    var color                   : String    = ""
    var name                    : String    = ""
    var manufacturer            : String    = ""
    var hasAirConditioning      : Bool      = true

    func describe() {
        print("\(paxCount), \(manufacturer)")
        Car.isTechProduct = true
        Car.description()
    }

    static func description(){

    }
}
```

6. Implement a child class called SUV.

```
class SUV : Car {
    var is4WheelDrive : Bool = true

    override func describe() {
        print("\(paxCount), \(manufacturer), \(is4WheelDrive)")
    }
}
```

7. Similarly implement other child classes as shown below.

```
class Sedan : Car {
    var isComfortable : Bool = true

    override func describe() {
        print("\(paxCount), \(manufacturer), \(isComfortable)")
    }
}

class LuxurySedan : Sedan {
    var hasNiceFittings : Bool = true
}
```

8. Create instance of the classes and see which properties are accessible and modifiable.

Exercise 1.6: Examine types of properties

Prerequisites

Exercise 1.1

Scope of Work

Explore the syntax for the different types of properties

Tools Required

Playgrounds

Outcome of the exercise

You should be comfortable with the different parts of a struct.

Steps

1. Create another Playground page. You can view the steps here.
2. Implement a struct as shown below.

```
struct Point {
    var x = 0.0

    init(x : Double, y : Double) {
        print("Point has been created")
    }

    var y = 0.0;
}
```

3. Modify the property x to have property observers.

```
struct Point {
    var x = 0.0 {
        willSet(newX) {
            print("New Value of X will be: \(newX)")
        }
        didSet
        {
            print("The old value was: \(oldValue)")
        }
    }

    init(x : Double, y : Double) {
        print("Point has been created")
    }

    var y = 0.0;
}
```

4. Implementing a lazy property is very easy. Just implement a property with the lazy keyword prefixed before it. Here is some code that shows you how to do that.

```
struct Frame {
    lazy var origin : Point = Point(x: 0.0, y: 0.0)
}

var myRect = Frame()
myRect.origin.x = 4.66
```

5. Next we will implement computed properties.

```
struct Size {
    var width  = 0.0
    var height = 0.0
}
```

6. Now implement a new type called Rect as shown below.

```
struct Rect {
    var origin  : Point     = Point(x: 0.0, y: 0.0)
    var size    : Size      = Size()

    var area    : Double {
        get {
            size.height * size.width
        }
    }

    mutating func newOrigin() {
        origin = Point(x: 0.0, y: 0.0)
    }
}
```

7. Add a new computed property called center as shown below.

```
struct Rect {
    var origin  : Point     = Point(x: 0.0, y: 0.0)
    var size    : Size      = Size()

    var center  : Point {
        get {
            return Point(x: origin.x + size.width/2.0,
                         y: origin.y + size.height/2.0)
        }
        set(newCenter){
            origin = Point(x: newCenter.x - size.width/2.0,
                           y: newCenter.y - size.height/2.0)
        }
    }

    var area    : Double {
        get {
            size.height * size.width
        }
    }

    mutating func newOrigin() {
        origin = Point(x: 0.0, y: 0.0)
    }
}
```

8. Create some instances of the type and test the different behaviours.

202327611082322384

Exercise 1.7: Examine Fail-able initialisers

Prerequisites

Exercise 1.1

Scope of Work

Explore the syntax for fail-able initialisers

Tools Required

Playgrounds

Outcome of the exercise

You should be comfortable with the different parts of a struct.

Steps

1. Create another Playground page. You can view the steps here.
2. Implement a class as shown below.

```
class Age {
    var personAge : UInt8?

    init(Given age : UInt8) {
        personAge = age
    }
}
```

3. Modify the init method as shown below.

```
class Age {
    var personAge : UInt8?

    init?(Given age : UInt8) {
        //if
        /*if age > 10
         {

         }*/
        guard age > 10
        else {
            print("Invalid age")
            return nil
        }
        personAge = age
    }
}
```

4. Try to create an instance which causes the init method to fail.

Exercise 1.8: Explore Generic types and generic functions

Prerequisites

Exercise 1.1

Scope of Work

Explore the syntax for generic functions

Tools Required

Playgrounds

Outcome of the exercise

You should be comfortable with the different parts of a generic functions and generic types.

Steps

1. Create another Playground page. You can view the steps here.
2. Implement a generic function as shown below.

```
func swapper<Element>(First first : inout Element, Second second : inout Element) {
    let temp = second
    second = first
    first = temp
}
```

3. Test the function with different types.

```
var num1 = 10
var num2 = 20
swapper(First: &num1, Second: &num2)

var str1 = "ABC"
var str2 = "EFG"
swapper(First: &str1, Second: &str2)
```

4. Now we will create a generic type.

```
class Number<Item> {
    var sourceData : Item?

    func description() {

    }
}
```

5. Try to create an instance of this type.

```
var intNumber : Number<Int> = Number<Int>()

intNumber.sourceData = 22
```

Exercise 1.9: Using associated type enums for exception handling

Prerequisites

Exercise 1.1

Scope of Work

Explore the syntax for associated type enums

Tools Required

Playgrounds

Outcome of the exercise

You should be comfortable with implementing exception handling logic.

Steps

1. Create another Playground page. You can view the steps here.
2. Implement the error types as shown below.

```
struct DataError {
    var code : Int
    var info : String
}

enum DataExceptions : Error {
    case FileNotFound(String)
    case FileCorrupted(DataError, String)
}
```

3. Implement a function that throws a function.

```
func processFile(status : Int) throws {
    guard status > 0
    else {
        throw DataExceptions.FileNotFound("The file was not present in the specified folder")
    }
}
```

4. Call the function and implement the exception handling logic.

```
do {
    try processFile(status: 0)
}
catch let error {
    print(error)
}
```

Exercise 1.10: Getting familiar with closures

Prerequisites

Exercise 1.1

Scope of Work

Explore the syntax for closures

Tools Required

Playgrounds

Outcome of the exercise

You should be comfortable with creating and using closures

Steps

1. Create another Playground page. You can view the steps here.
2. Implement the class as shown.

```
class Number {
    var value : Int?

    init() { }

    init(newValue : Int) {
        value = newValue
    }

    //: Closure being passed to a function
    func convertToString(converterClosure : (Int) -> String) -> String {
        if nil != self.value {
            return converterClosure(self.value!)
        }
        return "NaN"
    }
}
```

3. Create an instance of the class.

```
var myNumber : Number = Number(newValue: 33)
```

4. Now implement all the different variations of the closure one after the other.

```
//MARK: - Syntax 1
var secondClosure : (Int) -> String = {(input : Int) -> String in
    return "Data: \(input)"
}
let resp = myNumber.convertToString(converterClosure: secondClosure)
print(resp)

//MARK: - Syntax 2
myNumber.convertToString(converterClosure: {(input : Int) -> String in let str = "\(input)";
return str})

//MARK: - Syntax 3
myNumber.convertToString(converterClosure: {(input : Int) in let str = "\(input)"
    return str})

//MARK: - Syntax 4
myNumber.convertToString(converterClosure: {input in let str = "\(input)"
    return str})

//MARK: - Syntax 5
myNumber.convertToString(converterClosure: { return "\($0)" })

//MARK: - Syntax 6
myNumber.convertToString(converterClosure: {"\($0)"})

//MARK: - Syntax 7
myNumber.convertToString{"\($0)"}
```

4
Chapter 4: Design patterns

In this chapter the reader will get familiar with the different options available to developers when it comes to designing code.

Note
You will be performing all the exercises in separate playground pages. Please follow the instructions carefully. You are free to try all this code in a single playground too.

Exercise 4.1: Getting familiar with Protocols

Prerequisites

Should be familiar with Swift programming

Scope of Work

Learn about the different aspects of protocols

Tools Required

Xcode 14.x

Playground

Outcome of the exercise

You should be comfortable with protocols and their use.

Steps

1. Create a new playground or create a new playground page and continue with the playground from an earlier chapter.
2. At the top of the file import the Foundation framework.

```
import Foundation
```

3. Next we will see how to declare a protocol.

```
//:#### Declaring Protocols
protocol SomeRequiredMethods
{
    func description() -> String

    func specializedDescription() -> String
}
```

4. Now we will see how we can adopt protocols.

```
class Number<Item> : SomeRequiredMethods { //conform, follow, agree to abide by
    var value : Item?

    init(newItem : Item) {
        value = newItem
    }

    func description() -> String {
        if let retval = value
        {
            return "\(retval)"
        }
        return ""
    }

    func specializedDescription() -> String {
        if let retval = value {
            return "\(retval)"
        }
        return ""
    }
}
```

5. Try to create objects of Number and then run the code.

Exercise 4.2: Advanced Protocol concepts: Protocols as type

Prerequisites

Should be familiar with Swift programming

Scope of Work

Learn about the advanced aspects of protocols

Tools Required

Xcode 14.x

Playground

Outcome of the exercise

You should be comfortable with protocols oriented programming and using protocols as type.

Steps

1. Create a new playground or create a new playground page and continue with the playground from an earlier exercise.
2. Import foundation framework.
3. Declare 2 empty protocols.

```
//:#### Protocol Composition
protocol Searchable {

}

protocol Listable {

}
```

4. Declare an array type by composing the above two protocols as the internal type of the array.

```
var complexTypeArray : [Searchable & Listable] = [Searchable & Listable]()
```

5. Declare classes, create objects of those classes and try appending them to the array.

```
class C : Searchable, Listable {

}

class D : Searchable {

}

complexTypeArray.append(C())
//complexTypeArray.append(D())
```

Exercise 4.3: Advanced Protocol concepts: Associatedtype

Prerequisites

Should be familiar with Swift programming

Scope of Work

Learn about the advanced aspects of protocols

Tools Required

Xcode 14.x

Playground

Outcome of the exercise

You should be comfortable with associatedtype.

Steps

1. Create a new playground or create a new playground page and continue with the playground from an earlier exercise.
2. Import foundation framework.
3. Declare a protocol with an `associatedtype` as shown below.

```
//: Associated Type
protocol Mathematical
{
    associatedtype Item

    func logarithm(of number : Item) -> Item

    func sine(of number : Item) -> Item

    func cosine(of number : Item) -> Item

    func tangent(of number : Item) -> Item
}
```

4. Declare a class the adopts this protocol.

```
class MathOperations : Mathematical {
    typealias Item = Double

    func logarithm(of number : Item) -> Item {
        return log(number)
    }

    func sine(of number : Item) -> Item {
        return sin(number)
    }

    func cosine(of number : Item) -> Item {
        return cos(number)
    }

    func tangent(of number : Item) -> Item {
        return tan(number)
    }
}
```

Exercise 4.4: Advanced Protocol concepts: Conditional conformance

Prerequisites

Should be familiar with Swift programming

Scope of Work

Learn about the advanced aspects of protocols

Tools Required

Xcode 14.x

Playground

Outcome of the exercise

You should be comfortable with conditional conformance of protocols.

Steps

1. Create a new playground or create a new playground page and continue with the playground from an earlier exercise.
2. Import foundation framework.
3. Declare a simple implementation of a generic array.

```
//:#### Conditional conformance to protocols
class DemoArray<Item> {
    var internalArray : [Item] = [Item]()

    func insert(at index : Int, newValue value : Item) {
        internalArray.insert(value, at: index)
    }

    func append(NewValue value : Item) {
        internalArray.append(value)
    }

    var size : Int {
        return internalArray.count
    }

    subscript (index : Int) -> Item {
        get {
            return internalArray[index]
        }
        set(newValue) {
            internalArray[index] = newValue
        }
    }
}
```

4. Extend the class while conditionally conforming to the protocol.

```
extension DemoArray : Equatable where Item : Equatable {
```

5. Provide the implementation for the == operator.

```
/**
      This function implements the overload for `==` operator
      - important: This function does not validate data. The type on left hand side and right hand side of the operator must
match.
      - returns: `Bool`.
      - requires: iOS 11 or later
      - parameter lhs: value on LHS of operator
      - parameter rhs: value on RHS of operator
      - Since: iOS 11
      - author: Arun Patwardhan
      - copyright: Copyright (c) Amaranthine 2015 - version: 1.0*/
    @available(iOS, introduced: 11.0, message: "== operator")
    static func ==(lhs : DemoArray, rhs : DemoArray) -> Bool {
        if lhs.size == rhs.size && lhs[0] == rhs[0] {
            return true
        }
        return false
    }
}
```

6. Test it by creating an array of Ints and an array of non equatable types.

```
struct A {

}

var aArr : DemoArray<A> = DemoArray<A>()
var bArr : DemoArray<A> = DemoArray<A>()

var intArr : DemoArray<Int> = DemoArray<Int>()
var intArr2 : DemoArray<Int> = DemoArray<Int>()

if intArr == intArr2 {

}
```

A
Appendix A: Xcode

Common tasks that need to be performed with Xcode are found here.

202327611082322384

A1: Creating playground pages

Steps

1. Open Xcode.
2. File > New > Playground. This will create a playground file.
3. Click File > New > Playground Page. To create a new playground page. The first playground page and the playground itself will share the same code space.
4. Create subsequent playground pages by repeating step number 3.
5. Add the following code to incorporate previous and next hyperlinks to navigate all the pages.

```
//: [Previous](@previous)
```

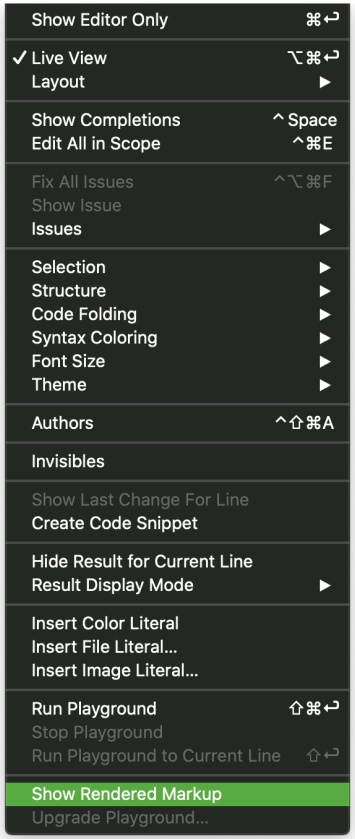
```
//: [Next](@next)
```

6. To get the rendered version of the formatted comments read article in appendix A on showing rendered markup.

A2: Rendering formatted comments in playground

Steps

1. Open Xcode.
2. From the menu bar, click on Editor > Show rendered markup.



3. Toggle this option to see the raw markup.

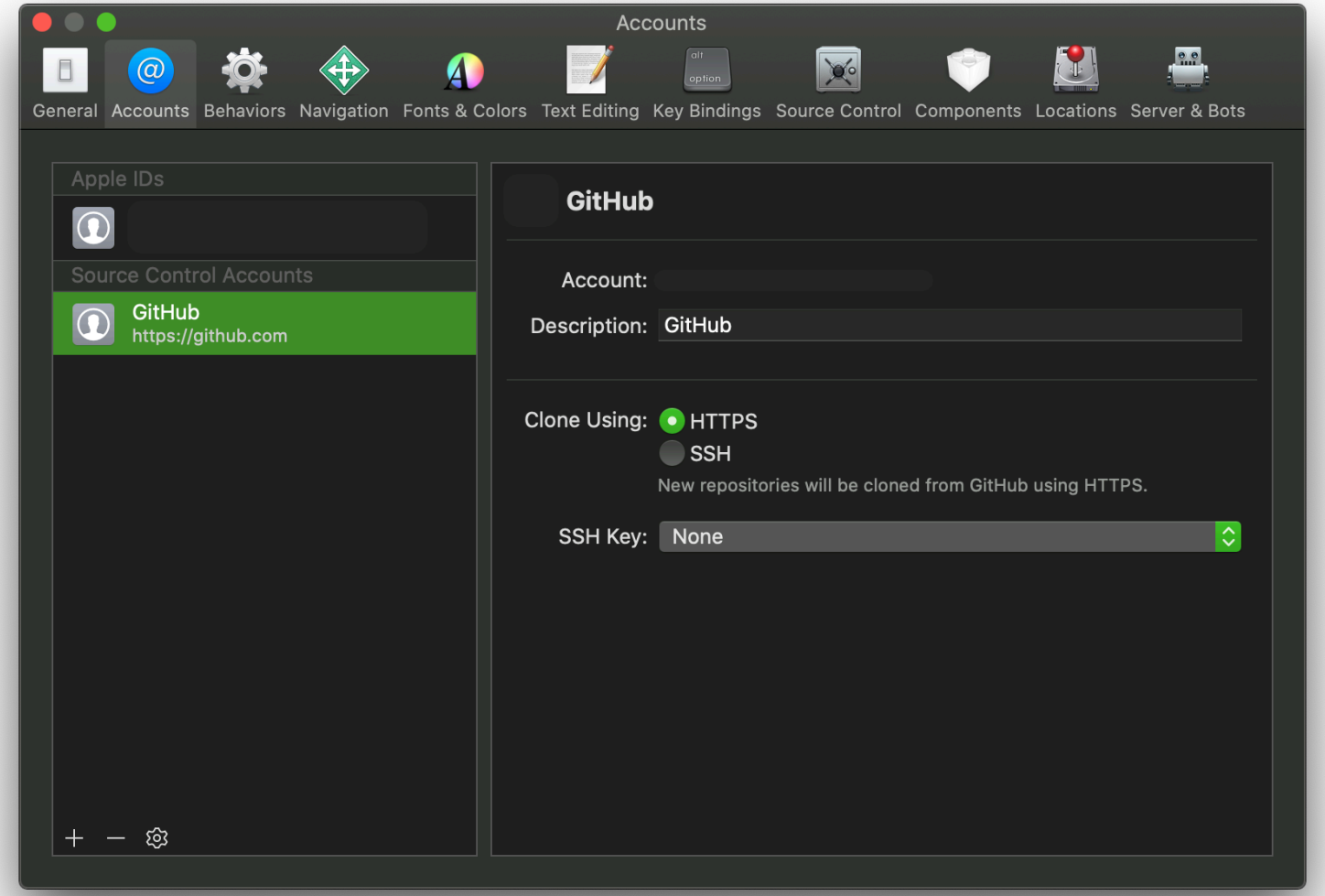
A3: Configuring Github on Xcode

You will need to ensure that you have a GitHub account available with you.

Steps

1. Open Xcode.
2. Click Xcode > Preferences.
3. Select Accounts.
4. Click on ‘+’ and select Github from the dropdown.
5. Login with your credentials.

6. You have now successfully configured GitHub on your computer.



A4: Erasing persistent stores on the simulator

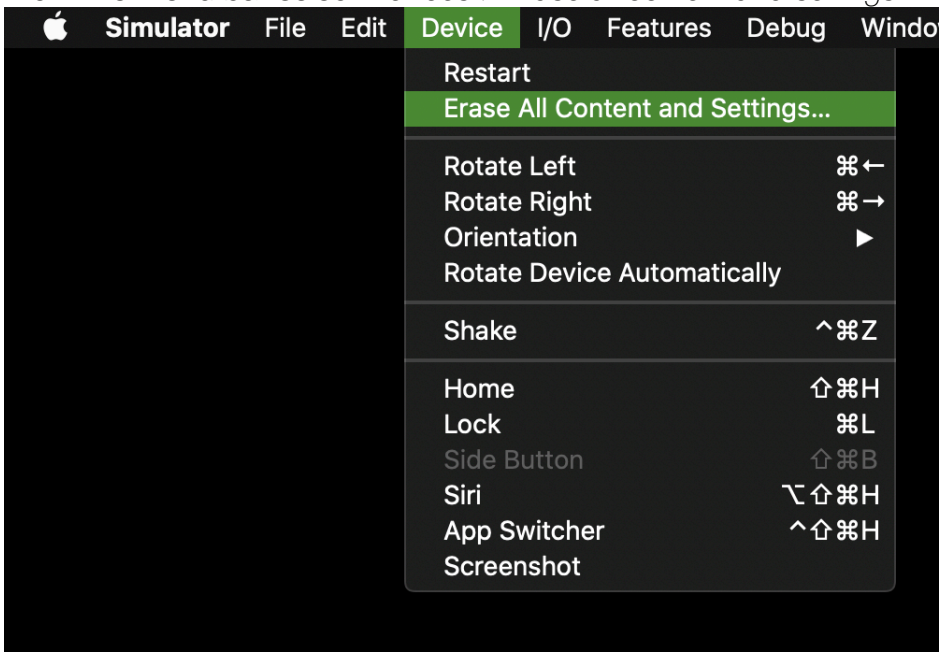
This is something you would have to do often when you are testing different persistent store solutions on the simulator.

Warning

This will wipe out all the apps and their content on the simulator. You will have to reload the apps all over again.

Steps

1. Open Simulator.
2. From the menu bar select *Devices > Erase all content and settings.*



3. Wait for the simulator to restart.

B
Appendix B: Third party tools

Common tasks that need to be performed with Xcode are found here.

B1: Installing & using Cocoapods

Steps

1. Open Terminal.
2. Run the following command:
   `sudo gem install Cocoapods`
3. Wait for Cocoapods to install.
4. To use them we first need an existing Xcode project. Take any project that you have or create a new one just to test cocoapods.
5. Quit Xcode.
6. Back in terminal, navigate to the folder where your Xcode project is. So supposing you created your project on the desktop then navigate to:
   `cd ~/Desktop/ProjectName/`
7. Your xcodeproj file would be located in here.
8. Run the command to initialise a pod.
   pod init
9. Change the generated PodFile by listing out the pods you wish to use.
10. Then run the command to install the pods.
    pod install[2]
11. Wait for the pods to install.
12. Once completed you are ready to start using them in your code.[3]
13. Open the xcworkspace file. Do not open the xcodeproj file.

---

[2] if you are running this on an Apple Silicon computer run the command **`sudo arch -x86_64 gem install ffi`** to install ruby. Then run the same command with the x86 architecture: **`arch -x86_64 pod install`**

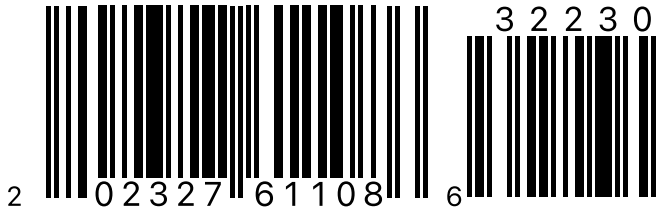
[3] You will have to check for the appropriate version numbers.

Contact

For any queries, corrections at the following

Website	www.amaranthine.in/contact-us
Email	arun@amaranthine.co.in
Facebook	https://www.facebook.com/amaranthinelabs
LinkedIn	https://www.linkedin.com/company/9410642?trk=tyah&trkInfo=clickedVertical%3Acompany%2CclickedEntityId%3A9410642%2Cidx%3A1-1-1%2CtarId%3A1438164427255%2Ctas%3Aamaranthine
Twitter	https://twitter.com/amaranthineTech
Youtube	https://www.youtube.com/channel/UC127UHd8V7bxPQYnd9QrN8w
Flipboard	https://flipboard.com/@AmaranthineTech
Instagram	https://www.instagram.com/amaranthinetech/

202327611082322384



Arun Patwardhan
arun@amaranthine.co.in
www.amaranthine.in