**Namespacing in Swift**

**Which scope are you talking about?**

You can have multiple functions that have the same name. There are some situations where you would probably actually want to do that.

This article explores Namespacing in Swift.

Photo by [Jon Tyson](https://unsplash.com/@jontyson?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText)on [Unsplash](https://unsplash.com/s/photos/name?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText)

Difficulty: Beginner | Easy| **Normal**| Challenging

**Prerequisites:**

* Be able to produce a “Hello, World!” iOS application (guide [HERE](https://medium.com/@stevenpcurtis.sc/your-first-swift-application-without-a-mac-79598ad839f8))
* Some knowledge of encapsulation (guide [HERE](https://medium.com/@stevenpcurtis.sc/data-encapsulation-in-swift-66adfb77dc1b))
* Knowledge of structs and classes (guide [HERE](https://medium.com/@stevenpcurtis.sc/classes-enums-or-structures-how-to-choose-your-swift-type-f33b4b76230e))
* Knowledge of Access Control is useful when Frameworks are invovled later in the article (guide [HERE](https://medium.com/swift-coding/access-control-in-swift-71228704654a))

**Terminology**

Frameworks: A way of sharing a package of code with Apps, team members or further afield

Module: A framework or application built and shipped as a single unit that can be imported by another module with Swift’s import keyword. Equivalent to node modules, packages, gems or jars in other languages

Namespace: A named region of a program used to group variables, types and methods

Struct: An object defined in Swift, using pass by value semantics

Target: A development target contains a set of source files that are compiled into a module or test suite

**Advantages of Namespaces**

Using Namespacescan give us the following advantages:

* Preventing name collisions
* Provides some form of data encapsulation

So now let us get stuck in learning about Namespacesin Swift

**Implicit namespacing**

Namespaceis not on a file bases, rather they are per target.

This, in effect means that we don’t need a Moduleprefix, so if we want to use the Integer type we are fine!

let myNum: Int = 4

which saves us from specifying that we are using Swift’s version

let myNum: Swift.Int = 4

now, in the same file what if we create our own rather silly **Int**type

struct Int {  
 var int: Swift.Int  
}

Now **Int**will refer to *our***Int**, that is the conflict is resolved as the

let myNum: Int = Int(int: 4)

So in a view controller this would look like the following:

class ViewController: UIViewController {  
 struct Int {  
 var int: Swift.int  
 }

override func viewDidLoad() {  
 super.viewDidLoad()  
 let myNum: Int = Int(int: 3)  
 print ("\(myNum)")  
 }  
}

**Namespacing with a Framework**

Namespacewith a Frameworkmeans that we need to make the initializer public across modules

struct Int {  
 var int: Swift.Int  
 public init(int: Swift.Int) {  
 self.int = int  
 }  
}

This can then be accessed from the main view controller if the Frameworkis imported, that is:

import UIKit  
import IntFrameworkOne

class ViewController: UIViewController {  
 override func viewdidLoad() {  
 super.viewDidLoad()  
 let myNum: Int = Int(int: 4)  
 print (myNum)  
 }  
}

therefore **Int**is taken from the Framework.

**Namespacing with Two Frameworks**

Namespacewith two Frameworksthat contain the same **Int**.

So at the top of the view controller we import both Frameworksand then proceed as before

import UIKit  
import IntFrameworkOne  
import IntFrameworkTwo

class ViewController: UIViewController {  
 override func viewdidLoad() {  
 super.viewDidLoad()  
 let myNum: Int = Int(int: 4)  
 print (myNum)  
 }  
}

This gives an error:

***‘Int’ is ambiguous for type lookup in this context***

which, of course, makes perfect sense.

So we can ***specify****which*Frameworkthat we mean

import UIKit  
import IntFrameworkOne  
import IntFrameworkTwo

class ViewController: UIViewController {  
 override func viewdidLoad() {  
 super.viewDidLoad()  
 let myNum: IntFrameworkOne.Int = Int(int: 4)  
 print (myNum)  
 }  
}

This works nicely!

**Conclusion:**

Namespacing is a way that **scope**is managed in your project. It is certainly worth getting to grips with, particularly if you have multiple Modulesor Frameworksthat you need to manage.

This article should begin to help you out and give you a grip on the challenges that will face you as you expand your code across files and more.

**Repo Link:**

Here is a repo link to help you out with Namespacing. Nice.

[**stevencurtis/Namespacing**  
*Contribute to stevencurtis/Namespacing development by creating an account on GitHub.*github.com](https://github.com/stevencurtis/Namespacing)

**Extend your knowledge**

* You can read about using local frameworks in the form of CocoaPods [HERE](https://medium.com/@stevenpcurtis.sc/using-local-pods-f4eb138314f9)

**The Twitter contact:**

Any questions? You can get in touch with me [here](https://twitter.com/profile)