Swift: An essential guide

Get up to speed with the programming language, Swift, and get coding quickly, **Mihalis Tsoukalos** motors through the basics.

Our expert

Mihalis Tsoukalos is a DBA, programmer, mathematician and Unix admin. In his spare time he likes to write technical articles and fight crime.

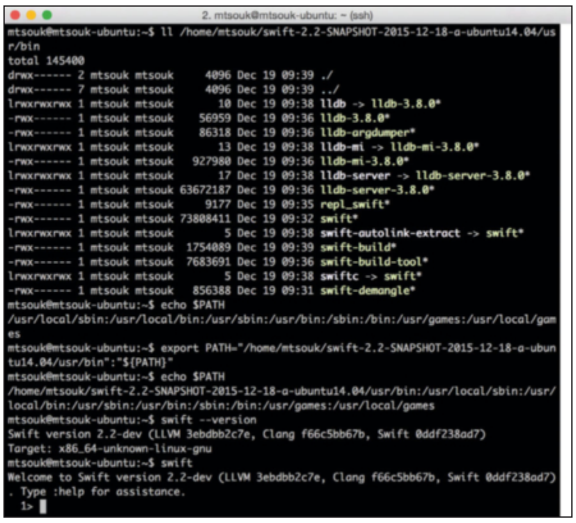
The biggest surprise of the Apple Worldwide Developer Conference 2014 (WWDC) was the announcement of Swift, a new programming language that will – eventually – replace Apple Objective-C. The biggest surprise of WWDC 2015 was the announcement that Swift was to become an open source project. This tutorial is a gentle introduction to Swift and its various features through a few small but complete code examples.

It's most likely, you will find the installation process the most difficult part of this tutorial. Apple doesn't provide packages for every Linux distribution (distro). Currently, only Ubuntu 14.04 and 15.10 are officially supported. If you have a different Linux distro, the installation process will likely be more challenging. For an Ubuntu 14.04 Linux system the installation process is:

```
$ sudo apt-get install clang libicu-dev
$ wget https://swift.org/builds/ubuntu1404/swift-2.2-SNAPSHOT-2015-12-18-a/swift-2.2-SNAPSHOT-2015-12-18-a-ubuntu14.04.tar.gz
$ wget -q -O - https://swift.org/keys/all-keys.asc | gpg --import -
$ gpg --keyserver hkp://pool.sks-keyservers.net --refresh-keys Swift
```

Quick tip

Parentheses for if statements are optional, because Swift considers that sometimes they add unnecessary noise to code. Swift also doesn't have **.h** files because you don't need them.



```
mtsouk@mtsouk-ubuntu:~$ ll /home/mtsouk/swift-2.2-SNAPSHOT-2015-12-18-a-ubuntu14.04/us
r/bin
total 145400
drwx------ 2 mtsouk mtsouk     4096 Dec 19 09:39 ./
drwx------ 7 mtsouk mtsouk     4096 Dec 19 09:39 ../
lrwxrwxrwx 1 mtsouk mtsouk       10 Dec 19 09:38 lldb -> lldb-3.8.0*
-rwx------ 1 mtsouk mtsouk    56959 Dec 19 09:36 lldb-3.8.0*
-rwx------ 1 mtsouk mtsouk    86318 Dec 19 09:36 lldb-argdumper*
lrwxrwxrwx 1 mtsouk mtsouk       13 Dec 19 09:38 lldb-mi -> lldb-mi-3.8.0*
-rwx------ 1 mtsouk mtsouk   927980 Dec 19 09:36 lldb-mi-3.8.0*
lrwxrwxrwx 1 mtsouk mtsouk       17 Dec 19 09:38 lldb-server -> lldb-server-3.8.0*
-rwx------ 1 mtsouk mtsouk 63672187 Dec 19 09:36 lldb-server-3.8.0*
-rwx------ 1 mtsouk mtsouk     9177 Dec 19 09:35 repl_swift*
-rwx------ 1 mtsouk mtsouk 73808411 Dec 19 09:32 swift*
lrwxrwxrwx 1 mtsouk mtsouk        5 Dec 19 09:38 swift-autolink-extract -> swift*
-rwx------ 1 mtsouk mtsouk  1754089 Dec 19 09:39 swift-build*
-rwx------ 1 mtsouk mtsouk  7683691 Dec 19 09:36 swift-build-tool*
lrwxrwxrwx 1 mtsouk mtsouk        5 Dec 19 09:38 swiftc -> swift*
-rwx------ 1 mtsouk mtsouk   856388 Dec 19 09:31 swift-demangle*
mtsouk@mtsouk-ubuntu:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/gam
es
mtsouk@mtsouk-ubuntu:~$ export PATH="/home/mtsouk/swift-2.2-SNAPSHOT-2015-12-18-a-ubun
tu14.04/usr/bin":"${PATH}"
mtsouk@mtsouk-ubuntu:~$ echo $PATH
/home/mtsouk/swift-2.2-SNAPSHOT-2015-12-18-a-ubuntu14.04/usr/bin:/usr/local/sbin:/usr/
local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
mtsouk@mtsouk-ubuntu:~$ swift --version
Swift version 2.2-dev (LLVM 3ebdbb2c7e, Clang f66c5bb67b, Swift 0ddf238ad7)
Target: x86_64-unknown-linux-gnu
mtsouk@mtsouk-ubuntu:~$ swift
Welcome to Swift version 2.2-dev (LLVM 3ebdbb2c7e, Clang f66c5bb67b, Swift 0ddf238ad7)
. Type :help for assistance.
  1> 
```

The contents of the bin directory of the Swift installation.

```
$ wget https://swift.org/builds/ubuntu1404/swift-2.2-SNAPSHOT-2015-12-18-a/swift-2.2-SNAPSHOT-2015-12-18-a-ubuntu14.04.tar.gz.sig
$ gpg --verify swift-2.2-SNAPSHOT-2015-12-18-a-ubuntu14.04.tar.gz
```

The first command installs some required packages in case they're not already installed. The second command downloads the latest development snapshot. The third imports PGP keys and should only be executed once on each Linux system. The fourth command downloads the latest keys and the fifth downloads the required signature files. The final command verifies that the archive that you've downloaded is intact. Bear in mind that the last command might produce a harmless warning message that you can ignore provided that you downloaded all files from a trusted source (eg **https://swift.org**).

If everything is fine so far, extract the archive with `tar xzf swift-2.2-SNAPSHOT-2015-12-18-a-ubuntu14.04.tar.gz`. This command creates a new directory that you should put in your current PATH variable:

```
$ export PATH="/home/mtsouk/swift-2.2-SNAPSHOT-2015-12-18-a-ubuntu14.04/usr/bin":"${PATH}"
```

Note: adjust the `/home/mtsouk/swift-2.2-SNAPSHOT-2015-12-18-a-ubuntu14.04` path in order to match your system, username and Swift directory. You can find your version of Swift by executing `swift --version`. (The screenshot (above) shows the contents of the **bin** directory of the Swift installation.)

Other Swift characteristics

Swift values are never implicitly converted to another type which makes your code less buggy. You should always use curly brackets (or braces) { and } even for blocks of code that only have one line. As we mention in the tutorial, the most powerful control statement in Swift is the switch statement. As Swift also supports functional programming, a Swift function can return another function as its value! Additionally, you can use any Unicode character as a variable name. So, eg you can say:

```
let ϖ = 3.14159
```

Swift understands when a variable that was declared as var never changes its value and prints the following warning:

```
bubbleSort.swift:7:21: warning: variable 'temp' was never mutated; consider changing to 'let' constant
```

Swift also supports the while and the repeat-while statements which is similar to the do-while statement used in C.

If you execute Swift without any command line arguments you will be presented with the Swift REPL. The REPL is the perfect place to learn new things because it's an interactive shell that allows you to read, evaluate and print the results of any Swift code you enter. All Swift functionality is available from the REPL. Additionally, if you press the tab key, the REPL will present you with a list of available completions based on the text you've already typed. Let's start by presenting you with the "Hello World!" program written in Swift:

```
print("Hello, World!")
```

If you are familiar with C, C++ or Objective-C, you'll likely make a few observations. First, there are no semicolons that denote the end of a command. Swift doesn't use semicolons unless you want to put two commands on the same line. Second, there's no main() function which should be a great surprise for you. Last, there are no #include or #import statements; however, this only happens because the "Hello World" program is small and doesn't need any.

You can execute the "Hello World!" program as follows:

```
$ swift hw.swift
Hello, World!
$ swiftc hw.swift
$ ls -l hw
-rwxrwxr-x 1 mtsouk mtsouk 13676 Dec 24 18:19 hw*
$ ./hw
Hello, World!
$ cat hw.script
#!/home/mtsouk/swift-2.2-SNAPSHOT-2015-12-18-a-ubuntu14.04/usr/bin/swift
print("Hello World!")
$ chmod 755 hw.script
$ ./hw.script
Hello World!
```

As you can see there are three different ways to execute Swift code. The first executes Swift code from the command line without creating an executable file. The second method uses `swiftc` to compile **hw.swift** and generate an executable binary file that's automatically named **hw** based on the filename of the file with the Swift code. The third way shows how to create and execute a Swift script (**hw.script**) – this is analogous to a Unix shell script. If you can successfully run the Hello World program then you're ready to continue.

Apple has included the Swift Package manager and a Build System in the Swift package which offers a handy way for compiling Swift projects. We'll use the **hw.swift** file as an example. First, you'll need to create a new directory that will include all project files with `$ mkdir hw and $ cd hw`.

Next, you must create a file called **Package.swift** that will be the manifest file of the project – this is mandatory. If the manifest file is empty, the package manager will create the new project using the default options. As you are using the defaults, the package manager expects to find all Swift code inside a directory called **Sources** which you must also create:

```
$ touch Package.swift
$ mkdir Sources
```

By default, the package manager will search for a file named **main.swift** and compile it, so copy **hw.swift** inside the **Sources** directory but name it **main.swift** and build the project as follows:

```
$ cp ~/hw.swift Sources/main.swift
$ swift build
```

The final product will be available inside the **.build/debug** directory. This happens because the package manager considers that your first executable will be used for debugging. The name of the executable file will be the same as the name of the project. (See the screenshot (*below*) for the full list of created files and directories that are created for a project.) The rest of the tutorial will show important features of the language so get ready to write some Swift code!

Constants and variables

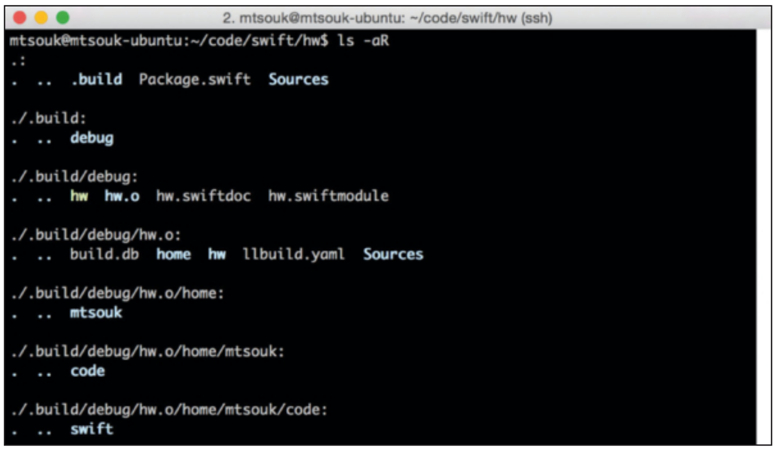
Swift requires you to declare either constants or variables. The code (*below*) defines a variable and a constant inside the Swift REPL:

```
1> var youCanChangeMe = 12
youCanChangeMe: Int = 12
 2> let youCannotChangeMe = 12
youCannotChangeMe: Int = 12
```

You should use the `var` keyword to declare a mutable variable and `let` to declare an immutable variable. Swift will generate a very descriptive error message if you try to change the value of `youCannotChangeMe` (*as you can see below*). »

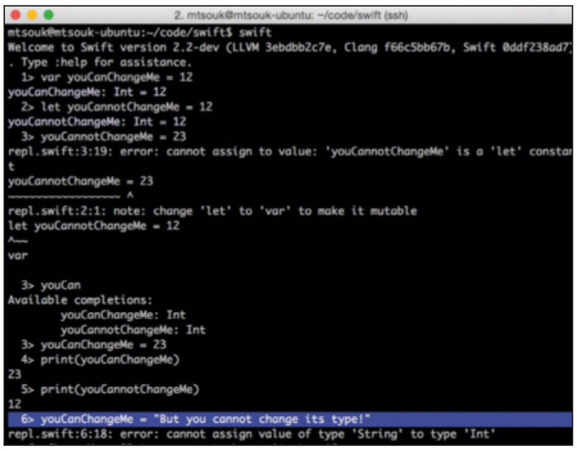
Quick tip

Two good books about Swift are *Swift Programming: The Big Nerd Ranch Guide* by Matthew Mathias, John Gallagher and *The Swift Apprentice: Beginning Programming with Swift 2* by Janie Clayton, Alexis Gallagher, Matt Galloway, Eli Ganem, Erik Kerber and Ben Morrow.



```
2. mtsouk@mtsouk-ubuntu: ~/code/swift/hw (ssh)
mtsouk@mtsouk-ubuntu:~/code/swift/hw$ ls -aR
.:
.  ..  .build  Package.swift  Sources

./.build:
.  ..  debug

./.build/debug:
.  ..  hw  hw.o  hw.swiftdoc  hw.swiftmodule

./.build/debug/hw.o:
.  ..  build.db  home  hw  llbuild.yaml  Sources

./.build/debug/hw.o/home:
.  ..  mtsouk

./.build/debug/hw.o/home/mtsouk:
.  ..  code

./.build/debug/hw.o/home/mtsouk/code:
.  ..  swift
```

› **The Swift Package Manager, which is included in the Swift package, creates a plethora of files and directories when building a project.**



```
2. mtsouk@mtsouk-ubuntu: ~/code/swift (ssh)
mtsouk@mtsouk-ubuntu:~/code/swift$ swift
Welcome to Swift version 2.2-dev (LLVM 3ebdbb2c7e, Clang f66c5bb67b, Swift 0ddf238ad7
. Type :help for assistance.
  1> var youCanChangeMe = 12
youCanChangeMe: Int = 12
  2> let youCannotChangeMe = 12
youCannotChangeMe: Int = 12
  3> youCannotChangeMe = 23
repl.swift:3:19: error: cannot assign to value: 'youCannotChangeMe' is a 'let' constar
t
youCannotChangeMe = 23
~~~~~~~~~~~~~~~~~ ^
repl.swift:2:1: note: change 'let' to 'var' to make it mutable
let youCannotChangeMe = 12
^~~
var

  3> youCan
Available completions:
        youCanChangeMe: Int
        youCannotChangeMe: Int
  3> youCanChangeMe = 23
  4> print(youCanChangeMe)
23
  5> print(youCannotChangeMe)
12
  6> youCanChangeMe = "But you cannot change its type!"
repl.swift:6:18: error: cannot assign value of type 'String' to type 'Int'
```

› You can't change the value of a let variable – only the value of a variable declared with var is allowed to be changed. Also, you can't change the type of a variable!

» Additionally, you're not allowed to change the type of a variable even if it was declared with var. The Swift language is clever enough to automatically understand the type of a variable. However, you can still declare the type of a variable yourself using the following notation:

```
var youCanChangeMe: Int = 123
```

The Swift code (*below*) defines a recursive function that will enable you to calculate integers that belong to the Fibonacci sequence:

```
func fibo(number: Int) -> Int {
  if number == 0 {
    return number
  }
  if number == 1 {
    return number
  }
    return (fibo(number - 1) + fibo(number - 2))
}
```

The declaration of the `fibo()` function shows that it needs an integer as its input, which is assigned to the number variable and that it also returns an integer. Should you wish to calculate the first 15 numbers of the Fibonacci sequence, you should use the following for loop:

```
for i in 0...15 {
        print("Fibonacci number \(i) is: \(fibo(i))")
}
```

The use of the `\()` notation inside `print()` allows the programmer to include a variable or the return value of a function inside a string. If you are familiar with C, you can see that you don't need to specify the type of the variable that's going to be printed.

Alternatively, you can write the same for loop as follows:

```
let loop = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
for i in loop {
    print("Fibonacci number \(i) is: \(fibo(i))")
}
```

The naive last example shows that you can iterate over the items of an array using a for loop. Despite the new style of the Swift for loop, you can still use the old style because it's still valid, eg: `for i = 0; i<=15; i++ { }`.

However, future Swift versions might remove the C-style for loop so you'd be advised to use the new for loop. If you don't want to include the right bound in your for loop, you should write it as `for i in 0..<5 { }`. If, however, you want to use a specific step, you can write the for loop as `/for i in 2. stride(to: 10, by: 2) { }`.

The previous for loop only returns 2, 4, 6 and 8 using the `stride()` function. The `stride()` function returns a sequence of values of any 'strideable' type and can also be used for getting a reverse range. Should you wish to get a reverse range without using stride, you should use the reverse() function as follows:

```
for i in (1...5).reverse() { }
```

Please bear in mind that the loop is invalid and will generate an error message when you try to execute it:

```
for i in 5...1 { }
```

As you can see from the last example, `stride()` also works for doubles!

Quick tip

Swift also supports Structures, Classes, Enumerations, Protocols, Generics, Error Handling and Functional Programming and can be used as a scripting language.

Swift functions

Swift functions are very powerful and support many innovative features including variadic parameters, in-out parameters and the ability to return tuples. Variadic parameters represent zero or more parameters of a given type. We illustrate this in the next example (**variadic.swift**):

```
func Sum(integers: Int...) -> Int {
        var total = 0
        for number in integers {
                total += number
        }
        return total
}

var total = Sum(10, 30, 50, -17, 9, 110, -130, 17, 100)
print("The total is \(total)")
```

The sum function converts the variadic variable integers into a constant array named integers with Int elements that are accessed using a for loop.

In-out function parameters should remind you of the 'call by reference' functionality found in C. This is used when you want the changes you make inside a function to persist after the function call has ended. The following Swift code illustrates in-out function parameters (**inAndOut.swift**):

```
func minMax(inout min: Int, inout max: Int) {
        if min > max {
                let temp = min
                min = max
                max = temp
        }
}
var myMin = 100
var myMax = -10
print("(Before) Min: \(myMin) and Max: \(myMax)")
minMax(&myMin, max: &myMax)
print("(And After) Min: \(myMin) and Max: \(myMax)")
```

The **inAndOut.swift** code produces the following output:

```
(Before) Min: 100 and Max: -10
(And After) Min: -10 and Max: 100
```

Please note that Swift requires that you write the second argument label only and not both of them when calling `minMax()`. Therefore, the next two calls of `minMax()` will not be valid Swift code:

```
minMax(&myMin, &myMax)
minMax(min: &myMin, max: &myMax)
```

The error message produced by the first line of code would be the following:

```
inAndOut.swift:13:7: error: missing argument label 'max:' in call
minMax(&myMin, &myMax)
    ^
           max:
```

The second call of the `minMax()` function will generate the following error message:

```
inAndOut.swift:13:7: error: extraneous argument label 'min:' in call
minMax(min: &myMin, max: &myMax)
```

The way you should call a function with more than one argument, and the error messages produced might look a little strange but this is how Swift currently works.

The use of tuples provides a nice way of grouping values of any type. They allow you to pass or receive multiple values easily and allow functions to return multiple values without using a structure or a separate object.

```
import Glibc
func minMax(array: [Int]) -> (min: Int, max: Int) {
...
   return (min, max)
}

var randomArray = [Int]()
// Put 10 random numbers in the array
for i in 1...10 {
           var number = random()
           randomArray.append(number)
}

let myMM = minMax(randomArray)
// Alternatively
var (myMin, myMax) = minMax(randomArray)

print("Max is \(myMM.max) and Min is \(myMM.min)")
print("Max is \(myMax) and Min is \(myMin)")
```

As you can see there are two ways to get and use a tuple: as a single variable (`myMM`) or as separate variables (`myMin` and `myMax`). If you don't know the total number of fields a tuple has then the first method is better. The declaration of `minMax()` shows that it returns two values and that it requires its input to be an array of integers with no specific size. The key point in the `minMax()` implementation is the return call at the end of it. The `import Glibc` command allows you to use existing C functions in your Swift code and is the equivalent of the import Darwin command used on Mac OS X. Please see the **tuplesRandom.swift** file for the full implementation of the minMax() function.

The most powerful control statement in Swift is switch, as it can use regular expressions in its branches and even match tuples to make your life easier. It's better suited to more complicated conditions and executes the appropriate block of code based on the first successful match. As it doesn't need a break statement, it avoids executing more than one case by mistake. Just don't forget that a switch statement must either cover all possible cases explicitly or have a default part for catching what was not covered so far. You also get the matched values and use them afterwards:

```
case (let x, let y): print("You are at \(x) and \(y)")
```

Swift optionals

Optionals is another new Swift feature that will help you reduce bugs in your code. The code (below) shows how they are used (**optionals.swift**) with a dictionary structure:

```
let Contacts = ["George": "123211212", "Georgia": "23211223"]
let isGeorgiaPresent: String? = Contacts["Georgia"]

if isGeorgiaPresent == nil {
           print("You have no contact named Georgia!")

}
else {
           let contactNumber = isGeorgiaPresent!
           print("The contact number of Georgia is \(contactNumber)")
}
```

The previous code allows you to find out whether a given key can be found in the dictionary and act accordingly. The question mark (`String?`) indicates the optionality of the value. The `nil` value works with any type and means 'there is no value present'. If you're sure that a variable has a value, you can use an exclamation mark (!) to get a value from an optional which is called 'force unwrapping'. In case you forget to put the '!', the output from the previous Swift code would be:

```
The contact number of Georgia is Optional("23211223")
```

As you can see, the presented output isn't particularly useful. So bear in mind that the unwrapping process – which is also called 'forcing the value out' – is mandatory.

This part will implement the Bubble sort algorithm in Swift. The source code of **bubbleSort.swift** is the following:

```
func bubbleSort(inout array: [Int]) {
   let elements = array.count - 1
   for var i=elements; i>=1; i-- {
      for var j=0; j<=i-1; j++ {
                         // Swap them if needed
         if array[j] > array[j+1] {
            let temp = array[j]
            array[j] = array[j+1]
            array[j+1] = temp
         } } } }
```

The bubbleSort function uses an in-out parameter for the input array so that changes to it inside the function will persist after the function call has ended. Note that the `array.count` variable holds the total number of elements in an array not the maximum index number.

And that's your lot. Keep programming in Swift to learn all its capabilities because Swift is here to stay! LXF

Next issue: Swift projects

Closures

Closures are similar to blocks in Objective-C and to lambdas in other programming languages. Strictly speaking closures are small self-contained blocks of functionality that can be passed around and used in your code. Closures look pretty much like a function declaration, and that's because functions in Swift are just named closures!

One important difference between closures and functions is that once a function has been defined, its name cannot be reassigned whereas the name of a closure can be reused so you cannot be sure that a named closure does the same thing throughout the entire program execution. Closures are used because they are more compact than regular functions especially when used in combination with other functions such as map.