SEE
TEXT
ONLY

Swift Programming Language

by Steve Wills

·Private·
·Fileprivate·

Swift is a new general purpose programming language from Apple that was announced at Apple's annual WWDC event in 2014 and released as open source in 2015 under version 2.0 of the Apache License. It is designed to replace Objective C, but to be more concise and safer, and has been. Swift is primarily targeted at iOS application development, but it is a complete general purpose systems programming language and can be used for many tasks. It is under rapid development and is portable to many operating systems. Swift fully supports Unicode. It's easy to learn, fun to use, and fast to run. Swift supports both imperative and object-oriented programming, as well as generics, extensions, try/throw/catch error handling, dynamic dispatch, extensible programming, and late binding.

Safety

Memory is managed automatically, so there's no need to manually allocate or free it, which avoids errors. As part of its design goal of making programming safer, Swift largely avoids exposing pointers to the developer, though it is possible to work with them where needed. It requires variables to be declared before use, avoiding issues with implicit declaration sometimes seen in scripting languages. Types may be inferred for the sake of conciseness or declared where needed for safety and clarity. Data types are strictly enforced for safety, but types can be cast easily for flexibility.

Five different access controls are supported for symbols (variables, functions, classes, etc.):

- Private - accessible only in the immediate scope
- Fileprivate - any code within the same file may access
- Internal - any code within the same module may access
- Public - accessible from any module
- Open - may be subclassed outside of the module (classes and methods only)

These access controls apply regardless of inheritance and allow explicit control of where code and data are used, avoiding surprises and difficult troubleshooting.

·Internal·
·Public·
·Open·

How to Build It on FreeBSD

First, ensure you have FreeBSD 11 and the latest FreeBSD ports tree. Then:

```
cd /usr/ports/lang/swift ; make install
```

Swift uses a custom version of llvm, clang, lldb, cmark, and llbuild, so the build will take some time. After this, you should be able to run the "swift" command and get an interactive prompt. You can also save Swift files and call the Swift interpreter on them.

What's Missing?

Swift can be interpreted or compiled, but currently only the interpreter works on FreeBSD. As of this writing, the lang/swift port is version 2.2.1, while the current release of Swift is 3.0.1. Work is underway to update the port and enable the compiler. Swift tutorials are often focused around writing iOS apps, but those require libraries that are part of iOS, which at this time are only available on iOS. Therefore, only the core Swift programming language is available on FreeBSD.

Hello, World!

As with any programming language, the first program we write in Swift is one that prints "Hello, World". In Swift, it's as simple as:

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

When entered into the interpreter, it looks like this:

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

Swift doesn't require semicolons at ends of lines, but does allow them:

```
(swift) print("Hello"); print("World!");
Hello
World!
```

This helps keep the code easy to read, while also allowing developers to be as concise as they want.

Variables and Constants

Variables must be declared before they are used:

```
(swift) var message = "Hello, World!"
// message : String = "Hello, World!"
(swift) print(message)
Hello, World!
```

Again, the goal of conciseness is served by allowing the developer to skip declaring the variable type since Swift has automatically determined that our message is a string type. Of course, where needed or desired, variable types may be declared:

```
(swift) var message2: String = "Hello"
// message2 : String = "Hello"
```

Whoops, we forgot part of our message. Let's add it:

```
(swift) // add rest of message
(swift) message2+=", World!"
(swift) print(message2)
Hello, World!
```

Once again, conciseness is served by allowing simple string concatenation.
We can also specify constants:

```
(swift) let message3 = "Hello"
// message3 : String = "Hello"
```

Which can't be modified:

```
    (swift) message3+=", World!"
    <REPL Input>:1:9: error: left side of mutating operator isn't mutable:'message3'
is a 'let' constant
    message3+=", World!"
    ~~~~~~~~^
    <REPL Input>:1:1: note: change 'let' to 'var' to make it mutable
    let message3 = "Hello"
    ^~~
    var
```

This serves the goal of safety by allowing the programmer to specify some values as read-only where necessary. The error message makes it easy to see where to change things if we want to make a variable read/write.

Types

Swift provides all C and Objective-C types, including Int, Double, Float, Bool, and string for textual data.

```
    (swift) var four: Float = 4
    // four : Float = 4.0
    (swift) var five: Double = 5
    // five : Double = 5.0
    (swift) let six: Float = 6
    // six : Float = 6.0
```

Once again, safety is served by enforcing variable type:

```
     (swift) let result = four + five
     <REPL Input>:1:19: error: binary operator '+' cannot be applied to operands of
type 'Float' and 'Double'
     let result = four + five
                  ~~~~ ^ ~~~~
     <REPL Input>:1:19: note: overloads for '+' exist with these partially matching
parameter lists: (Float, Float), (Double, Double)
     let result = four + five
                       ^
     (swift) let result = four + six
     // result : Float = 10.0
```

But ease and flexibility are served by allowing them to be cast easily:

```
     (swift) let result2 = four + Float(five)
     // result2 : Float = 9.0
```

Swift helps us read numbers more easily:

```
     (swift) let million = 1_000_000
     // million : Int = 1000000
     (swift) print(million)
     1000000
```

And once more, conciseness is served since Swift allows us to assign multiple variables at once:

```
(swift) var (foo, bar, baz) = (10, 100, 1000)
// (foo, bar, baz) : (Int, Int, Int) = (10, 100, 1000)
```

In addition, Swift provides Array, Set, Dictionary, Ranges, and Tuples:

```
(swift) let myarray: Array<String> = ["first", "second", "third"]
// myarray : Array<String> = ["first", "second", "third"]
(swift) let secondarray = [1: "Alice", 2: "Bob", 3: "Chuck"]
// secondarray : [Int : String] = [2: "Bob", 3: "Chuck", 1: "Alice"]
(swift) let People: Dictionary<Int, String> = [1: "Alice", 2: "Bob", 3: "Chuck"]
// People : Dictionary<Int, String> = [2: "Bob", 3: "Chuck", 1: "Alice"]
(swift) let range = 0...5
// range : Range<Int> = Range(0..<6)
(swift) let range2 = 0..<10
// range2 : Range<Int> = Range(0..<10)
```

Swift also introduces the innovative concept of Optionals, variables where values may or may not be present:

```
(swift) var daytime: Bool? = true
// daytime : Bool? = Optional(true)
(swift) var mayContainNumber = "404"
// mayContainNumber : String = "404"
(swift) var actualNumber = Int(mayContainNumber)
// actualNumber : Int? = Optional(404)
```

Swift implied that our actualNumber may or may not contain a number, rather than forcing us to specify it. We use the "!" operator to get the value:

```
(swift) print("actualNumber has an integer value of \(actualNumber!).")
actualNumber has an integer value of 404.
```

However, we must be safe and check that the value exists before using it:

```
(swift) actualNumber = nil
(swift) print("actualNumber has an integer value of \(actualNumber!).")
fatal error: unexpectedly found nil while unwrapping an Optional value
```

So, we must write:

```
(swift) if actualNumber != nil {
            print("actualNumber has an integer value of \(actualNumber!).")
        }
(swift) actualNumber = Int(mayContainNumber)
(swift) if actualNumber != nil {
            print("actualNumber has an integer value of \(actualNumber!).")
        }
actualNumber has an integer value of 404.
```

Type Safety

Unlike C, the assignment operator does not return a value, so this produces a nice error message and once again keeps us safe from easy typos:

```
    (swift) if foo = bar {
                print("fail")
            }
```

CODE CONTINUES ON PAGE 14

CONTINUED FROM PAGE 13

```
  <REPL Input>:1:8: error: use of '=' in a boolean context, did you mean '=='?
  if foo = bar {
     ~~~ ^ ~~~
         ==
```

And also, helpfully suggests the necessary change.
Similarly, integers cannot be used as booleans:

```
  (swift) let i = 1
  // i : Int = 1
  (swift) if i {
            print("foo")
          }
<REPL Input>:1:4: error: type 'Int' does not conform to protocol 'BooleanType'
if i {
   ^
```

Along with the requirement that variables must be declared before use, these requirements eliminate several classes of common mistakes.

Control Flow

Swift has the usual control flow mechanisms, "if" and "switch" for conditionals, and "for" and "while" loops:

```
var value = 10
// Note {} are required for "if" and "while", even with just one statement
if value > 0 {
  print("true")
}

// Note lack of break statements, cases do not fall through by default, but
// can with "fallthrough"

var weekday = "Thursday"
switch weekday {
case "Monday":
  print("They call it stormy Monday")
case "Friday":
  print("TGIF")
default:
  print("Is it Friday yet?")
}

// Note use of Range and "in"
for value in 1...10 {
  print("\(value) * = \(value * 9)")
}

while value > 0 {
  print("Not yet")
  value -= 1
}
```

Functions

Swift uses named arguments to functions, which makes the code easier to read and maintain, though in Swift 2.x the first name must be left off. Functions may or may not return a value. Take this example code (on next page):

```
// function which greets user
// returns the greeting
func hello(user: String) -> String {
  let result = "Hello, " + user + "!" // result not modified, so constant
  return result
}

/* function which greets user a second time
   (comments may be of either style) */
func helloAgain(user: String) -> String {
  return "Hello again, " + user + "!"
}

// function which takes additional bool arg
func hello(user: String, seen: Bool) -> String {
  if seen {
    return helloAgain(user)
  } else {
    return hello(user)
  }
}

// greet multiple users, doesn't return a value
func helloAll(users: [String]) {
  // Ensure we were given users, otherwise next line would error
  guard users.count > 0 else { return }
  users.forEach { user in
   print(hello(user, seen: false))
   }
}

let users = ["Alice", "Bob", "Chuck"]
var seenUsers = ["Alice": true, "Bob": false]
seenUsers["Chuck"] = false
var nousers: Array<String> = Array()
helloAll(users)
print(hello("Alice", seen: true))
helloAll(nousers)
```

Save it to a file called multihello.swift and run it:

```
$ swift multihello.swift
Hello, Alice!
Hello, Bob!
Hello, Chuck!
Hello again, Alice!
```

Variadic functions are supported via "...":

```
func sum(values: Int...) -> Int {
    var sum = 0
    for value in values {
         sum += value
    }
    return sum
}
sum(10, 20, 30)
```

Function nesting is supported:

```
func returnValue() -> Int {
   var i = 1
   func add() {
        i += 1
   }
   add()
   add()
   add()
   return i
}
returnValue()

(Returns 4)
```

And since functions are a first class type, they may be returned from other functions:

```
func returnAdder() -> ((Int) -> Int) {
     func adder(number: Int) -> Int {
          return 1 + number
     }
     return adder
}
var adder = returnAdder()
adder(10)
```

RootBSD

Premier VPS Hosting

RootBSD has multiple datacenter locations,
and offers friendly, knowledgeable support staff.
Starting at just $20/mo you are granted access to the latest
FreeBSD, full Root Access, and Private Cloud options.

www.rootbsd.net

Classes

Swift is also object oriented. Classes can be declared like so:

```
  class Building {
     var numberOfFloors = 0
     init(numberOfFloors: Int) {
        self.numberOfFloors = numberOfFloors
     }
     func getDescription() -> String {
        return "A building with \(numberOfFloors) floors."
     }
 }

 class PaintedHouse: Building {
     var color: String

     init(floors: Int, color: String) {
        self.color = color
        super.init(numberOfFloors: floors)
     }
     override func getDescription() -> String {
         return "A \(color) building with \(numberOfFloors) floors."
     }
 }

 var house = Building(numberOfFloors: 2)
 var houseDescription = house.getDescription()

 let myHouse = PaintedHouse(floors: 3, color: "blue")
 print(myHouse.getDescription())
```

Finale

There's much more to Swift, and I encourage everyone to give it a try. There are many good online resources on learning Swift:

https://swift.org/
https://developer.apple.com/swift/
https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/GuidedTour.html

There are even several that allow you to run code online via web browser:

https://www.weheartswift.com/swift-sandbox/
https://swiftlang.ng.bluemix.net/#/repl

STEVE WILLS is a husband and father living in North Carolina. He is a FreeBSD ports committer with a focus on Ruby and other programming languages.