Swift

Aron Ockerse

CIS 343 01 - Structure of Programming Languages

Winter 2018

Development of Apple’s programming language titled Swift began in 2010, led by software developer Chris Lattner along with other Apple developers. The idea behind Swift was to create a compiled language that was more user friendly and was easily manageable by Apple itself. Prior to the release of Swift, all of Apple’s OS X and iOS operating systems were written in Objective-C. Although Objective-C was a powerful language it had its down sides. Objective-C’s biggest limitation was its inability to manage memory effectively. Apple found a workaround by implementing ARC within the software. ARC, which is short for Automatic Reference Counting is used in Objective-C and was continued to be used later in Swift. It is a memory management feature that adds code in at compile time in order to preserve the life of objects for as long as it is needed. As effective as this method was to preserve the longevity of Objective-C it still was only a superset of C and was very insecure to program in. With Swift Apple took everything that worked well with Objective-C and cleaned up the syntax immensely. Continuing the ARC memory management Objective-C did previously swift has been able to increase its compile time speeds tremendously. Swift is a multi-paradigm programming languages and its most notable paradigms being object oriented and functional programming. Much similar to Java and Python Swift can create objects in order to store any required data that can be later manipulated. What makes Swift a functional program is due to the fact that the objects it uses has function calls within it including recursion. These functions are dependent on input alone and are mapping the parameters to themselves. In 2015 Apple let Swift become fully open sourced to help further the development of the language with the community. Because of this many software developers were able to work with Apple’s software. Although Swift is limited to Apple it has seen a remarkable spike in iOS or Mac OS application development that have revolutionized the way people use their iphones and macbooks today.

Swift holds seven basic data types including: int, double, float, string, bool, array, and dictionaries. Examples of how some of these are written are as follows:

var numCars = 5; var carPrice = 100.00

let name = "Aron"

var carList ["Honda", "Ford", "Jeep]

var carColor ["Honda" : "Red", "Ford" : "Blue", "Jeep" : "Green"]

Swift uses what is called type inference which means the program can automatically deduce what data type the assigned variable is at compile time, but in order to reference a variable ‘var’ or ‘let’ must precede the variable name. Var is the standard way to declare any variable while let is a way to create a constant. Because Swift is statically typed it makes it easier to debug greatly increasing its readability as newcomers to the language are less likely to get confused over terms like integer and boolean. Also adding on to the example posted above is that there was only one semicolon typed. Semicolons are completely optional, however, if multiple data types are created on the same line it must be separated by a semicolon. All variable declarations are implicit when written unless any data type is casted then it is explicit.

//implicit

var a = 0

//explicit

var b : Double = 1.0

A strongly typed programming language is when a function or method is called the parameters that are passed in must match. Being statically typed Swift makes sure that all variables are correctly defined ahead of time.

func add(num1 : Int, num2 : Int) -> Int {

return num1 + num2

}

var a = 2

add(num1 : a, num2 : a)

Depending on what the programs task is there may be times to the user may need to change the data type of a variable with coercion. Much like Java to type coercion in Swift insert the desired data type in parenthesis before the variable.

var a = 1; var b = 2.5

var c = (Double) a + b

As of Swift’s latest version there are a total of four different types of language expressions those include: prefix expressions, binary expressions, primary expressions, and postfix expressions. A prefix expression is a combination of a prefix operator followed by the expression. An example of this can be seen used with the try expression.

a = try function()

The try is also a error handling expression which will end the program if anything happens throwing a runtime error. Binary expressions are the more general data type assignments. It will consist of a left argument a middle operator and a right argument.

var fruit = "apple"

Primary expressions are known as the building block expression which can be used by itself or in union with other expressions. The self expression is an example of this. Within method functions using self can be used to manipulate variables much similarly to Python.

var a = 0

func getNum(number : Int) {

self.number = number

}

Postfix expressions primarily include function calls and are more specifically what is going into the parameter.

func myName(name: String) -> String {  
 return name  
}  
myName(name: "Aron Ockerse")

These are the general standards and syntax when it comes to expressions in Swift, although the rules start to change when handling operators and precedence rules in if/else statements and for/while loops. In if statements Swift follows the more common way of comparing data types much like Java where it has: equals to (a == b), not equal to (a!=b), greater than (a>b), less than (a<b), greater than or equal to (a >= b), and less than or equal to (a<=b). So a simple if statement could look like the following:

if (a < b){

a += 1

}

Another key point to note in that if statement is the increment operation. In earlier versions of Swift and Objective-C as well users were able to write increment statements like a++ or a = a++. According to the creator Chris Lattner he believed writing a = a++ was not any more difficult than writing a += 1, and it became more confusing to newer programmers in readability. Lattner was also trying to deviate away from C and Objective-C syntax to further show off that Swift fixes a lot with what was wrong with the two languages. This can also be seen in the syntax of for loops in Swift.

for x in 1...5 {

print(x)

}

With syntax very similar to Python the above for loop would print variable x five times. The for in part of the for loop can also be used to loop through the elements of a list or dictionary. While loops follow the traditional syntax of while plus the condition and whatever statements within it.

while (x < 10){

x += 1

print(x)

}

Both the for and while represent different types of selection constructs the language has to offer. Case statements are also an alternative to loops and are formatted in the following way.

var answer

switch answer {

case 1:

print("yes")

}

Swift’s order of operation rules follow the same rules as most of the modern languages used today. Parentheses are called first followed by multiplication/division and then addition and subtraction. Functions and function calling techniques follow a similar format to java in but is different enough to stand out as its own language.

func output(word : String) -> String {

return word

}

All functions start with the func keyword and then the name of the method. If the function requires a parameter the name would be placed in the parenthesis followed by data type. A return type is optional depending on the specs of the program and would be written after the parameter with an arrow and the name of the data type.

func helloWorld(){

print("Hello World")

}

The helloWorld() function shows what a function would look like without a parameter or return type. Mentioned previously Swift has an exceptional handling tool in try, but also has other more effective ways to recover code. Swift is able to catch errors using do-catch method. The program will execute a function in try and catch will examine and check for any exceptions, if nothing is caught the do statement ends. To be able to have functions throw errors in the d0-catch methods the keyword throw needs to be places after a function declaration.

func randFunction() throw -> String

Swift has a unique way of handling namespacing in that it does it implicitly meaning it is all done within modules. By using the keyword import Swift can have access to the many different modules available.

Scopes in Swift are the time frame in which a variables, functions, or classes are attainable divided under two categories global and local scopes. Global scopes mean that any variables defined are accessible from any file under the same module while local scopes are subsets of these global scopes.

Object oriented programming plays a large role when it comes to programming in Swift. By typing class and a name a class template is created which is then can be used as an object to reference instances of that class.

class Person {

var name = String

var age = Int

}

Then with init the user can create multiple person objects of different people.

init (name : String, age : Int){

name = name

age = age

}

If an let aron = Person() were created helper functions within the class can be used to change the date if need be (aron.setAge(21)). Going more into detail with classes in Swift some classes can be turned into parent classes through interheritance.

Ever since Swift went open source its growth as a programming language is unbelievable. From how user friendly it is to how easy it is to do error check handling it is no wonder why it is among one of the top programming languages today.

Work Cited

Lattner, C. (n.d.). Apple/swift-evolution. Retrieved April 17, 2018, from https://github.com /apple/swift-evolution/blob/master/ proposals/0004-remove-pre-post-inc-decrement.md

Lifetime, Scope and Namespaces in Swift. (2016, January 21). Retrieved April 17, 2018, from https://andybargh.com/lifetime-scope-and-namespaces-in-swift/

Nyisztor, K. (2017, June 02). Why learn Swift? The creation and evolution of a new programming language. Retrieved April 17, 2018, from https://www.pluralsight.com/blog/software-development/swift-history

Swift From Scratch: An Introduction to Classes and Structures. (n.d.). Retrieved April 17, 2018, from https://code.tutsplus.com/ tutorials/swift-from-scratch-an-introduction-to-classes-and- structures--cms-23197

*The Swift Programming Language (Swift 4.1): About Swift*, 29 Mar. 2018,

developer.apple.com/library/content/documentation/Swift/Conceptual/Swift\_ Programming\_Language/index.html#//apple\_ref/doc/uid/TP40014097-CH3-ID0.

<https://stackoverflow.com/questions/13153140/what-are-the-major-differences-between-objective-c-c-and-c>

<https://www.infoworld.com/article/2920333/mobile-development/swift-vs-objective-c-10-reasons-the-future-favors-swift.html>

<https://www.computerworld.com/article/3219732/apple-ios/12-reasons-to-learn-apples-open-source-swift-language.html>

<http://www.bestprogramminglanguagefor.me/why-learn-swift>

<https://stackoverflow.com/questions/29924477/is-swift-a-dynamic-or-static-language>

<https://pchiusano.github.io/2016-09-15/static-vs-dynamic.html>

<https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/Expressions.html#//apple_ref/doc/uid/TP40014097-CH32-ID383>

<https://cocoacasts.com/what-is-the-difference-between-try-try-and-try>

<https://stackoverflow.com/questions/26247657/create-a-setter-only-in-swift>

<https://github.com/apple/swift-evolution/blob/master/proposals/0004-remove-pre-post-inc-decrement.md>

<https://andybargh.com/lifetime-scope-and-namespaces-in-swift/>

<https://code.tutsplus.com/tutorials/swift-from-scratch-an-introduction-to-classes-and-structures--cms-23197>