Swift

By: Ryan Holland
What is Swift?

- Apple released Swift as their new programming language on June 2nd of this year. Swift replaces current versions of objective-C used to program in the OSX and iOS environments.
- The reason for Swift was to improve existing compiler, debugger, and framework infrastructure. To simplify memory management with Automatic Reference Counting (ARC).
- It is built on the solid base of Foundation and Cocoa, and it has been modernized and standardized. Objective-C itself has evolved to support blocks, collection literals, and modules, enabling framework adoption of modern language technologies without disruption. Thanks to this groundwork, we can now introduce a new language for the future of Apple software development.
Swift is exclusively written in Xcode within OSX.
Mix and Match Languages

• Swift’s compatibility with Objective-C lets you create a project that contains files written in either language. You can use this feature, called mix and match, to write apps that have a mixed-language codebase.
Easy to use?

- Apple claims that Swift is more friendly to new programmers, also claiming that the language is just as enjoyable to learn as scripting languages. It supports a feature known as playgrounds, that allow programmers to make modifications on the fly, and see results immediately without the need to build the entire application first.
Basics

- Swift provides its own versions of all fundamental C and Objective-C types, including `Int` for integers, `Double` and `Float` for floating-point values, `Bool` for Boolean values, and `String` for textual data.
• Unlike most languages, Swift has a very small set of syntax that needs to be shown compared to other languages. An example of how simplistic the syntax is can be shown in one line.

   println("Hello, world!")

• This one line of code is a complete program. There is no need to import a certain library, string handling, or a main function. An added benefit being that you also do not need to end each statement with a semicolon.
Control Statements

- Control statements are easy to read and understand. To make conditionals the use of if and switch are used. Loops are created by for-in, for, while, and do-while. Optional values are represented by if and let together. If the value is missing, the nil expression is used. The use of while is to repeat a block of code until a change is read. The condition of a loop can be at the end of a statement to make sure that the entire loop is ran once.
Methods

• An instance method can be written within the opening and closing braces of the type it belongs to. An instance method has indirect access to all other instance methods and properties of the same type. An instance method can be called only on a specific instance of the type that it belongs to within the code. It cannot be called without existing anywhere else in the code.

```swift
class Counter {
    var count = 0

    func increment() {
        count++
    }

    func incrementBy(amount: Int) {
        count += amount
    }

    func reset() {
        count = 0
    }
}
```
Closures

- Closers are within standard programming language practices. Using `func` to declare functions. A function is called by listing arguments for that function within parentheses. An arrow is used to separate the parameter names and types (`->`) from a function's return type.

```swift
func backwards(s1: String, s2: String) -> Bool {
    return s1 > s2
}

var reversed = sorted(names, backwards)
```
Protocols

- Protocols are easily declared by simple use of the term protocol within the code. A protocol defines a blueprint of methods, properties, and other requirements that fit a particular task. A protocol can be adopted by a class, structure, or enumeration to provide an implementation of those requirements.

```swift
protocol SomeProtocol
{
    //definition of the protocol
}
```
Subclassing

- Subclassing is the act of basing a new class on an existing class within the code. The subclass inherits attributes from the existing class, which you can then redefine. New attributes can also be added to the subclass. To indicate that a subclass has a superclass, you can write the subclass name before the superclass name separated by a colon.

- Defining a subclass called Plane, with a superclass of Transportation.

class Plane: Transportation {
  var hasCargo = false
}
Readability

• Swift does not come as a perfect language, within it has its own faults. Yes, Swift is mostly different than others and cuts down on syntax and other issues that make beginning programmers complain. But to those experienced programmers, they are used to reading code that has basic closers to statements like a semicolon after each. In that respect, it shows them that the statement has finished. But since Swift does not have these semicolons, it can take a little adjustment from some programmers to read code in detail.
Writability

• The writability can get very tedious if not paying attention. Swift can really bring out the reasoning behind writing "clean code". Objects within swift make it easier to reuse existing code within applications, because you can create an object and inherit all of the code around it through attributes and methods. Which saves time from rewriting code every time you want to do something in an application.
Reliability

• Reliability can be a very strong point when using Swift, simply because it is Apple. Since Apple has such a closed ecosystem, it is easy for them to control code that is written and fix errors more easily the serverside. It makes things very easy for developers, because they can write and maintain code for such a smaller array of system specifications rather than writing code that will fit thousands of devices.

• An example would be writing and maintaining code for use of an app on an iPhone. Since iPhone specifications are rather specific, it is very reliable and easy to optimize. Rather than writing code for android that has to tailor to many devices, with many hardware configurations.
Cost

• The cost of using Swift can take a different approach than most. With all of the benefits of the reliability aspect of the code comes the downsides. Since Apple is such a closed ecosystem, only Apple made products can be legally used to run the Swift language. Swift code will not run in any other compiler outside of the OSX/iOS environments. So adding to the cost would be the hardware to run the software, in most cases OSX runs into issues when used alongside Linux/Windows operating systems. After the initial costs are out of the way for hardware, the software costs come into play. Nearly all of this code that is published has to pass inspection by the Apple team developers before deployment into an ecosystem. The time it takes for software to be approved can cost developers money, rather than other languages that are used in Linux/Windows that can be deployed instantly