Apple released their new programming language "Swift" on June 2nd of this year. Swift replaces current versions of objective-C used to program in the OSX and iOS environments. According to Apple, swift will make programming apps for these environments "more fun". Swift is also said to bring a significant boost in performance compared to the equivalent objective-C. Apple has not released an official document stating the reasoning behind the launch of Swift, but speculation is that it was a move to attract more developers to the iOS platform.

Swift development began in 2010 by Chris Lattner, but would later be aided by many other programmers. Swift employs ideas from many languages into one, not only from objective-C. These languages include Rust, Haskell, Ruby, Python, C#, CLU mainly but many others are also included. Apple says that most of the language is built for Cocoa and Cocoa Touch.

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. Apple controls most aspects of this language, and since they control so much of it, they keep a detailed records of information about the language on their developer site. Most information within this paper will be based upon those records.

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.

-Values-

Values are also easy to assign. To make a constant, let is used and for a variable var is used. Each type between a constant and variable must match one another, but do not have to be written, then are instead implied. When a value is created, the compiler reads the initial value of the integer that was first read and specifically knows its type automatically. Values inside of strings are very simple to write, all that is needed is a backslash before the parentheses. Arrays are like most languages, and use brackets with the elements inside of them.

-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 be called without existing anywhere else in the code. Apple iOS Developer Library shows a straightforward example of a simple counter class, which can be used to count the number of times an action occurs:
class Counter {
    var count = 0
    func increment() {
        count++
    }
    func incrementBy(amount: Int) {
        count += amount
    }
    func reset() {
        count = 0
    }
}

-Creating a class-

Creating a class as the normal approach similar to most other languages with the use of a `class` function declaration. If cleanup is needed within the code before an object is deallocated, a statement of `deinit` is used. Each subclass includes their own superclass name after their class name, which is separated by a colon. There is no requirement needed for any subclass to be attached to a standard root class. During class creation the compiler can also detect methods that include an `override` function, but do not actually override any methods that are a part of the superclass.

-Abstraction-

Staticred also explains abstraction rather simply, by showing that abstraction within Swift is used within a Model-View-Controller design pattern. This design pattern makes sure that the functions on the presentation layer do not read the structure of the data on the "model" level. An abstract function will exist within the Controller level that acts as a binding between the two, taking the View's request for data and passing it to the Model to retrieve. An example would be like this, which would call the `getVideo()` function inside of an app.

```swift
$video = new Video();
$video->id = 1850;
$video->getVideo();
```
-Closers-

Closers are within standard programming language practices. Using `func` to declare functions. A function is called by listing arguments for that function within parentheses. To separate the parameter names and types an arrow is used (->) to separate the name and types from a function's return type. Functions can also be nested within this language. Nested functions that are declared have direct access to variables that were declared in the outside function. Nested functions can be used if code seems to be too complex, or long to read. Apple's Swift programming library provides a function called `sorted`, which sorts an array of values of a known type, based on the output of a sorting closure that you provide in the code. One way you can provide a sorting close is to write a normal function of the correct type, and pass it in as the functions sorted second parameter like this:

```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. You can define protocols very easily to classes, structures, or enumerations like this:

```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. An example could be shown like this:

*Defining a subclass called Plane, with a superclass of Transportation:*

```swift
class Plane: Transportation {
    var hasCargo = false
}
```

**-Overview-**

Swift as an overall language makes heavy use of Object Oriented Programming. This is due to the enormous amount of users that are going to be using Swift, are going to be developing object oriented programs. These programs range from gaming apps to business software in OSX.

Evaluating Swift as a language is both easy and hard. It is easy in the fact that it is very easy to read due to the lack of certain syntax that seems to be repetitive in many other languages. A large portion of Swift code that I have evaluated the past month has had a trend of not using a lot of spacing. In many languages, such as Java, you will notice heavy use of spaces. Normally to organize code in a descending fashion. But Swift code is odd in the fact that it seems very organized, without needing the extra spaces or any descending type of code.

**-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. Or they can find themselves using closer statements or syntax that is not needed which will result in code errors.

**-Writability-**

The writability can get very tedious if not paying attention. Swift can really bring out the
reasoning behind writing "clean code". Much like staticred.com explains in their blog. 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. According to Ahmend Eid of the Huffington Post, Swift has very loose coding standards that a team must adhere to. It can be easy to write very efficiently yet unreadable of written badly.

-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 down sides. 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.
Apple also charges a yearly fee to developers to be in their program (typically only $100 a year), which pays for support alongside other things. When deploying code into the ecosystem, developers are able to set their own prices, receive monthly payments, no credit card fees, no hosting fees, and no marketing fees. Nearly all transactions are controlled through Apple which can make things easier for developers by just dealing with one entity. Apple also takes a 30% cut from all advertising on developer applications within their own ecosystem.

-Conclusion-

Overall this language is great, either for beginners or veterans. It has many helpful tools while developing, as well as a strong community support through forums or even by phone. Swift has one of the best communities to back it, and one of the most reliable platforms available. Swift does come at a cost, which is very steep for many. So many warn before deploying Swift and developing on the platform, to weigh out the costs and benefits before using it.
Appendix

Writing Values:
```swift
var myVariable = 36
myVariable = 48
let myConstant = 36
```

Creating arrays and dictionaries and accessing their elements within brackets:
```swift
var pcparts = ["Graphic card", "Ram", "SSD", "Motherboard"]
pcparts[1] = "power supply"

var stores = [
    "Walmart": "Groceries",
    "Best Buy": "Electronics",
]
stores["Books a Million"] = "Books"
```

A colon is used to write type annotation when you declare a constant or variable, to be clear about the kind of values for constant or variable can store. Use a colon after the constant or variable name, followed by a space, followed by the name of the type you want to use.

Control Flow:
```swift
let testscores = [88, 89, 93, 74, 98]
var passed = 0
for score in testscores {
    if score > 60 {
        passed += 1
    } else {
        passed += 0
    }
}
passed
```

Objects and Classes:
```swift
class Wheels {
    var numberOfSpokes = 0
    func simpleDescription() -> String {
        return "A wheel with \(numberOfSpokes) spokes."
    }
}
```
References


Introducing Swift

Ryan Holland
CSC415 Term Paper Final Draft
Dr. Lyle
Due: 11/4/14