Swift Research Paper

Philip Winchester
Independent Study
Professor James Lee

Swift is Apple’s new app development language. To use Swift the user has to have a Mac, or a Mac operating system, and Xcode. Xcode is Apple’s IDE that is designed around Swift and specifically OS X and iOS development with Swift. Swift has received some dubious opinions upon its’ release, because it is designed to be user friendly. Most notably Swift allows you to use emoji’s as variable names, which feels like a gimmick since it serves no practical purpose. However, once you start working with Swift you can see that the language is powerful and designed with very specific goals.

The main goal of Swift is to be a highly readable and writable language. It is a C-based language, which means that Swift is the spiritual successor to Objective-C. Objective-C was the old language used by Apple to develop its software and also used by the public for app development. Now Swift has taken over that mantle as Apple’s main development language. However, this means that many aspects of Xcode that work with Swift are still being developed. Also the online resources to help with Swift are scarce and the places you can learn Swift are equally slim. If you manage to find a good online resource and begin studying Swift you will find it an incredibly easy to learn language.

First off, Swift takes a page out of Python’s book and removes the need for semicolons at the end of every line (you can still use them if you want but they are unnecessary). Next, and again similarly to Python, variables do not need to be typed. Apple took this approach to make writing Swift easier, since once a variable is typed, which it does by inference, then the programmer will need to convert the variable if they

Philip Winchester
Independent Study
Professor James Lee

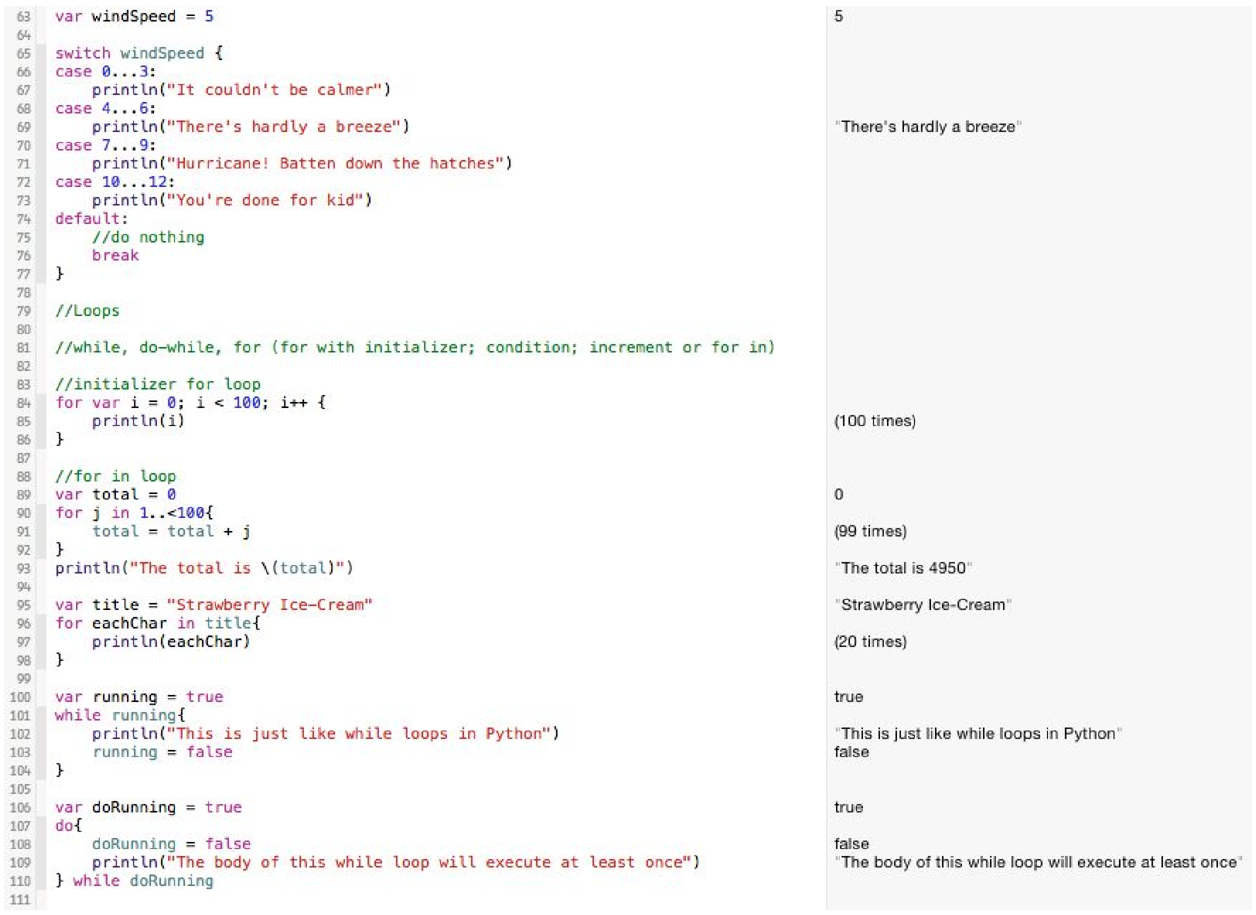
wish to change it or somehow use it outside the bounds of a certain type. This means that despite appearances, Swift is a strongly typed language. Swift also utilizes both for-in loops and for loops. Finally Swift allows the programmer to define variables as either “var” or “let”, which defines the variable as going to change or going to be constant, respectively. Swift encourages programmers to utilize constants wherever possible since it leads to safer code. The safety of Swift is one of its main underlying traits, variables are strongly typed and the use of constants is expected. Both of these ideas mean that programs are more consistent and less likely to confusing errors. These styles are not unique to Swift, but Swift implements them in a way that is easy to use and makes reading code simple.

Swift’s capabilities for classes, structs and, enums are also well developed. Classes have inheritance from superclasses and must have initializers. Structs are a more simplistic class that do not have inheritance and do not necessarily require an initializer. Finally enums in Swift are slightly different than other languages and allow the programmer to create a unique data type with strict restrictions. None of these are unique to Swift, but Swift has its own unique take on each of these types. Swift also has the usual set of data types (int, float, double, string, bool, etc.) as well as built-in array and dictionary data structures. All of this does not set Swift above any other language, but Swift does not try anything outside of the box with these systems and sticks to the well established styles used by many other languages.

Philip Winchester
Independent Study
Professor James Lee

Swift is a language that is intended to be used with apps primarily, but Xcode comes with a file type that allows programmers to test code easily. These are called “Playgrounds” and they take advantage of Xcodes real-time compiler and add console display line by line with the code.

Figure 1:



```
var windSpeed = 5                                                          5

switch windSpeed {
case 0...3:
    println("It couldn't be calmer")
case 4...6:
    println("There's hardly a breeze")                                     "There's hardly a breeze"
case 7...9:
    println("Hurricane! Batten down the hatches")
case 10...12:
    println("You're done for kid")
default:
    //do nothing
    break
}

//Loops

//while, do-while, for (for with initializer; condition; increment or for in)

//initializer for loop
for var i = 0; i < 100; i++ {
    println(i)                                                             (100 times)
}

//for in loop
var total = 0                                                              0
for j in 1..<100{
    total = total + j                                                      (99 times)
}
println("The total is \(total)")                                           "The total is 4950"

var title = "Strawberry Ice-Cream"                                         "Strawberry Ice-Cream"
for eachChar in title{
    println(eachChar)                                                      (20 times)
}

var running = true                                                         true
while running{
    println("This is just like while loops in Python")                     "This is just like while loops in Python"
    running = false                                                        false
}

var doRunning = true                                                       true
do{
    doRunning = false                                                      false
    println("The body of this while loop will execute at least once")      "The body of this while loop will execute at least once"
} while doRunning
```

Figure 1 is an example of a playground, which allows you to type code and Swift will display the number of times that the loops execute and what outputs will be displayed. In cases of more complicated code it is possible to get a more detailed view of each

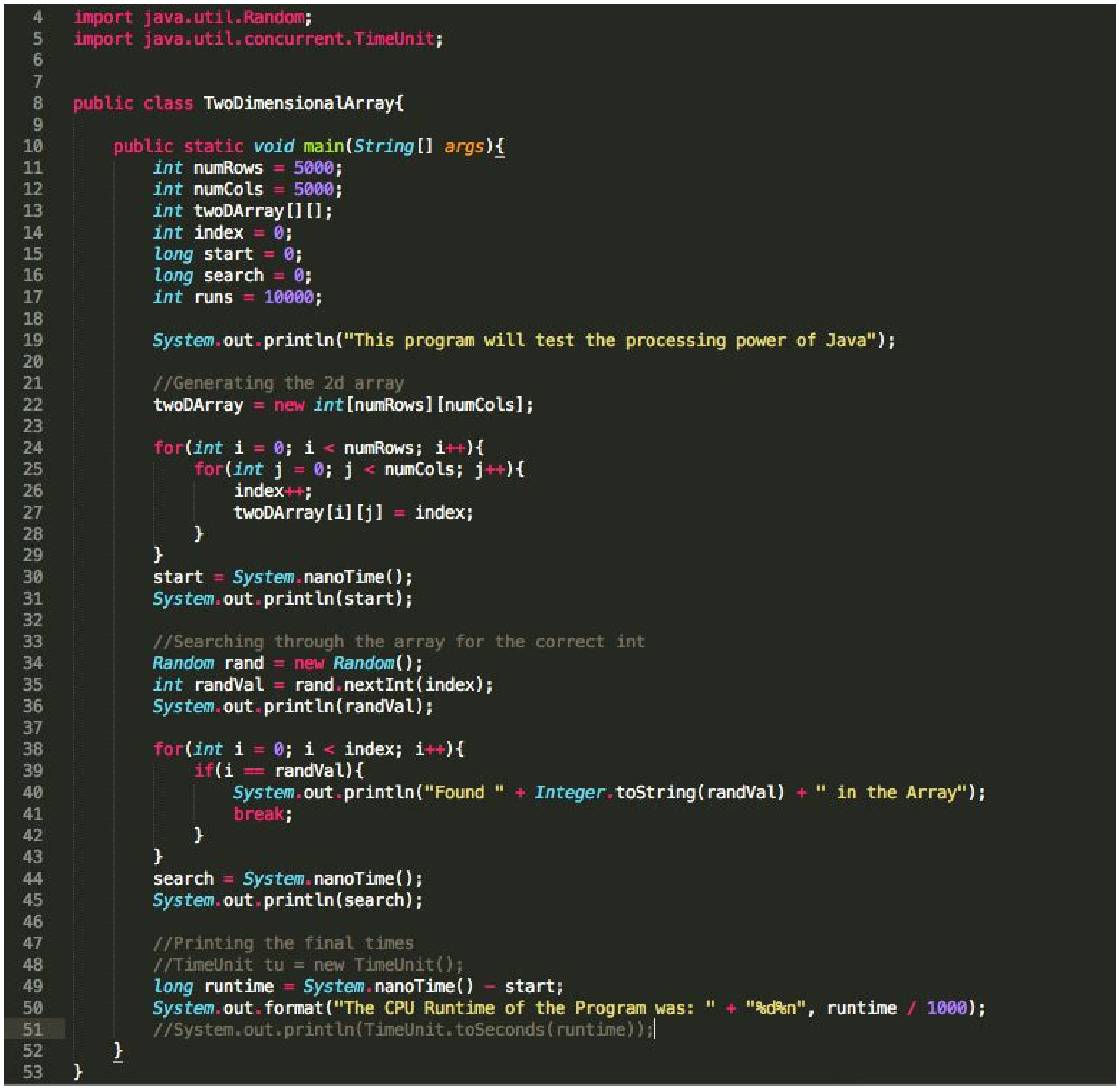
Philip Winchester
Independent Study
Professor James Lee

output, including the history of specific variables. There is also an execution slider to allow for easy debugging, so a programmer can follow their code bit by bit. Playgrounds are simple to set up and don't take any of the complicated set-up of a full project, but don't allow for good OS X or iOS testing. These playgrounds are a great way to test code, since running Swift files in the Mac Terminal is difficult (the command is "xcrun swift filename.swift" if you were curious). These playgrounds are a great way to learn the language and also to learn programming in generally since it allows for simple debugging and the output is directly next to each line of code. However, how does Swift's processing power stack up against other high level languages?

To test the power of Swift I decided to create a simple two dimensional array and attempt to index into it to find a specific value. I also did this same program in Python and Java to compare the speed at which they complete it compared to Swift. The screenshots of my code are below and I tried to keep these 3 programs as simple as possible. However, the goal was to calculate the processing time for each languages program and compare them. However, for the first version of these programs I had trouble getting them completely on even ground for fair comparison, so I will try and include all the difference I noticed to be as transparent as possible. For Java I couldn't find a clear way to convert it from nano-seconds to seconds, so the output is in nano-seconds. Roughly the Java program takes 2.3 seconds to complete. The Python program prints out out the value in seconds and comes out to 5.2-5.5 seconds depending on whether it is the first time running the program or not. So roughly twice as long as the Java program.

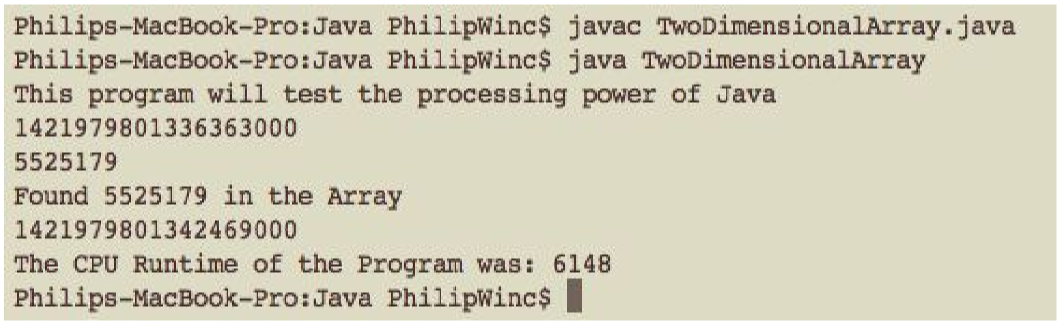
Finally the Swift file was much more difficult to work out. First I couldn't use a playground since I wanted to find the compile and run time of the program. Playground's compile in real-time so that wouldn't allow me to compare it against the Python and

Figure 2: Java Two Dimensional Array



```java
import java.util.Random;
import java.util.concurrent.TimeUnit;


public class TwoDimensionalArray{

    public static void main(String[] args){
        int numRows = 5000;
        int numCols = 5000;
        int twoDArray[][];
        int index = 0;
        long start = 0;
        long search = 0;
        int runs = 10000;

        System.out.println("This program will test the processing power of Java");

        //Generating the 2d array
        twoDArray = new int[numRows][numCols];

        for(int i = 0; i < numRows; i++){
            for(int j = 0; j < numCols; j++){
                index++;
                twoDArray[i][j] = index;
            }
        }
        start = System.nanoTime();
        System.out.println(start);

        //Searching through the array for the correct int
        Random rand = new Random();
        int randVal = rand.nextInt(index);
        System.out.println(randVal);

        for(int i = 0; i < index; i++){
            if(i == randVal){
                System.out.println("Found " + Integer.toString(randVal) + " in the Array");
                break;
            }
        }
        search = System.nanoTime();
        System.out.println(search);

        //Printing the final times
        //TimeUnit tu = new TimeUnit();
        long runtime = System.nanoTime() - start;
        System.out.format("The CPU Runtime of the Program was: " + "%d%n", runtime / 1000);
        //System.out.println(TimeUnit.toSeconds(runtime));
    }
}
```

Figure 3: Java Printout Values

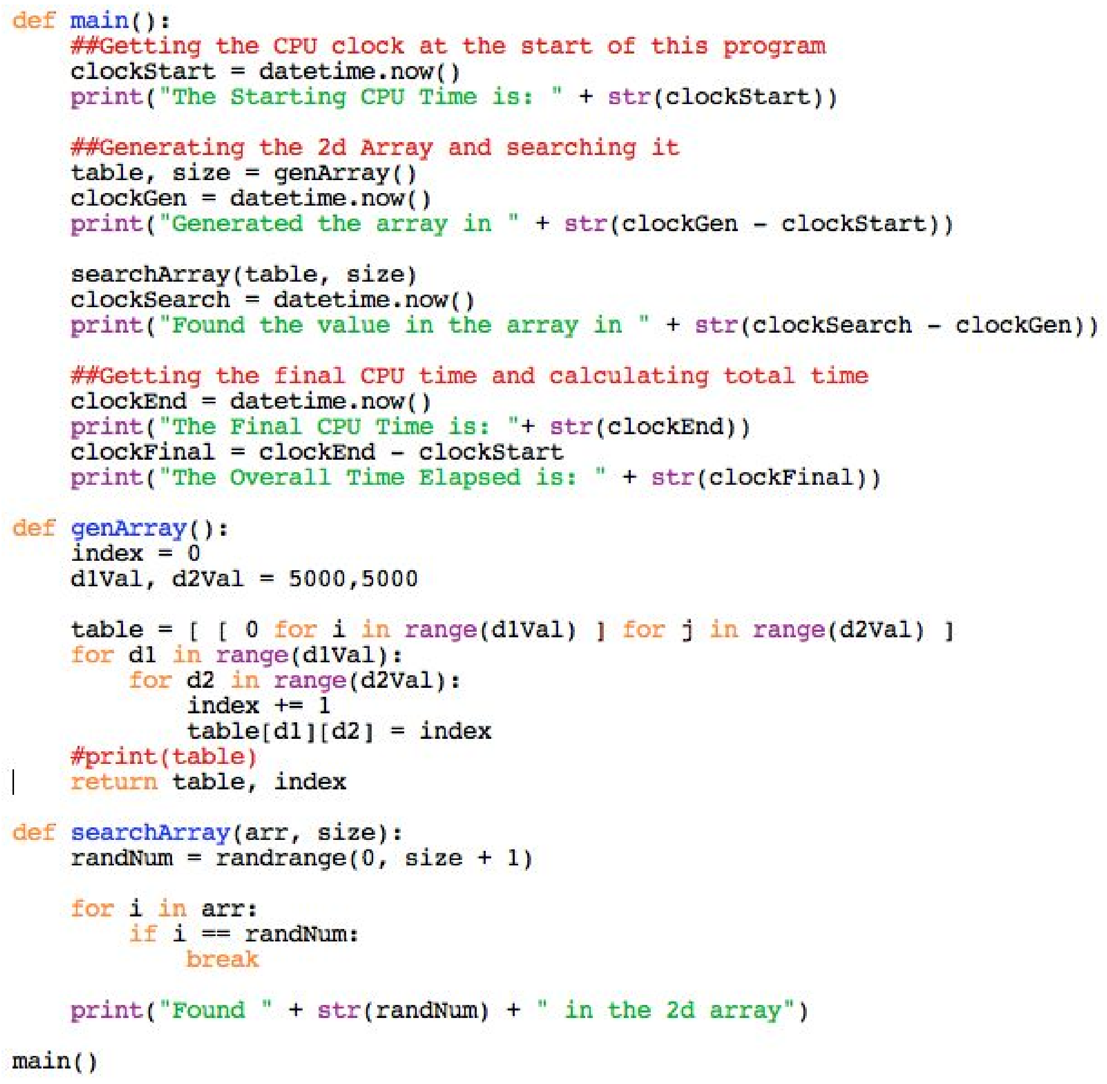


```
Philips-MacBook-Pro:Java PhilipWinc$ javac TwoDimensionalArray.java
Philips-MacBook-Pro:Java PhilipWinc$ java TwoDimensionalArray
This program will test the processing power of Java
1421979801336363000
5525179
Found 5525179 in the Array
1421979801342469000
The CPU Runtime of the Program was: 6148
Philips-MacBook-Pro:Java PhilipWinc$ 
```

java file. However, the more important problem is that I couldn't find a proper way to calculate the cpu processor time for Swift. This highlights the main issue with Swift, which is that it is such a new language that the online resources are difficult to find and especially on such a specific part of the language. The only help I could find was working with clock elements in Xcode for iOS. With all that clarified I ran the Swift file in Terminal and used a stopwatch to record the time. The test number for both Python and Java was a 5000 x 5000 array, but in Swift I couldn't get the completion time on that large an array. Compiling Swift in Terminal with a 5000 x 5000 array ended up with a 40 minutes elapsing before I stopped my program. I tried this same program, but in a playground and ran it for 2 hours before I had to stop. I was 4 million iterations through my nested loop to populate the array, before I had to stop my program. Finally I ran a 500 x 500 program in the playground, which finished it's task in roughly 3-4 minutes. Swift is a language designed with specific purposes in mind, and those purposes do not include processing large data structures for information. Swift is designed for app development, but lacks the adaptability that many other high level languages possess.

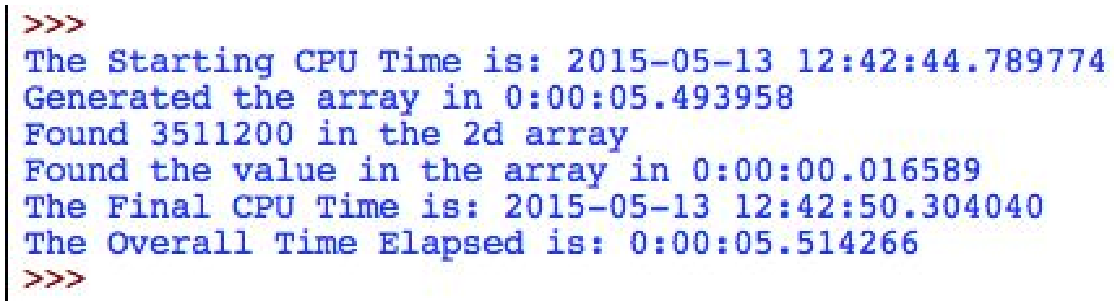
Philip Winchester
Independent Study
Professor James Lee

Figure 4: Python Two Dimensional Array



```python
def main():
    ##Getting the CPU clock at the start of this program
    clockStart = datetime.now()
    print("The Starting CPU Time is: " + str(clockStart))

    ##Generating the 2d Array and searching it
    table, size = genArray()
    clockGen = datetime.now()
    print("Generated the array in " + str(clockGen - clockStart))

    searchArray(table, size)
    clockSearch = datetime.now()
    print("Found the value in the array in " + str(clockSearch - clockGen))

    ##Getting the final CPU time and calculating total time
    clockEnd = datetime.now()
    print("The Final CPU Time is: "+ str(clockEnd))
    clockFinal = clockEnd - clockStart
    print("The Overall Time Elapsed is: " + str(clockFinal))

def genArray():
    index = 0
    d1Val, d2Val = 5000,5000

    table = [ [ 0 for i in range(d1Val) ] for j in range(d2Val) ]
    for d1 in range(d1Val):
        for d2 in range(d2Val):
            index += 1
            table[d1][d2] = index
    #print(table)
    return table, index

def searchArray(arr, size):
    randNum = randrange(0, size + 1)

    for i in arr:
        if i == randNum:
            break

    print("Found " + str(randNum) + " in the 2d array")

main()
```

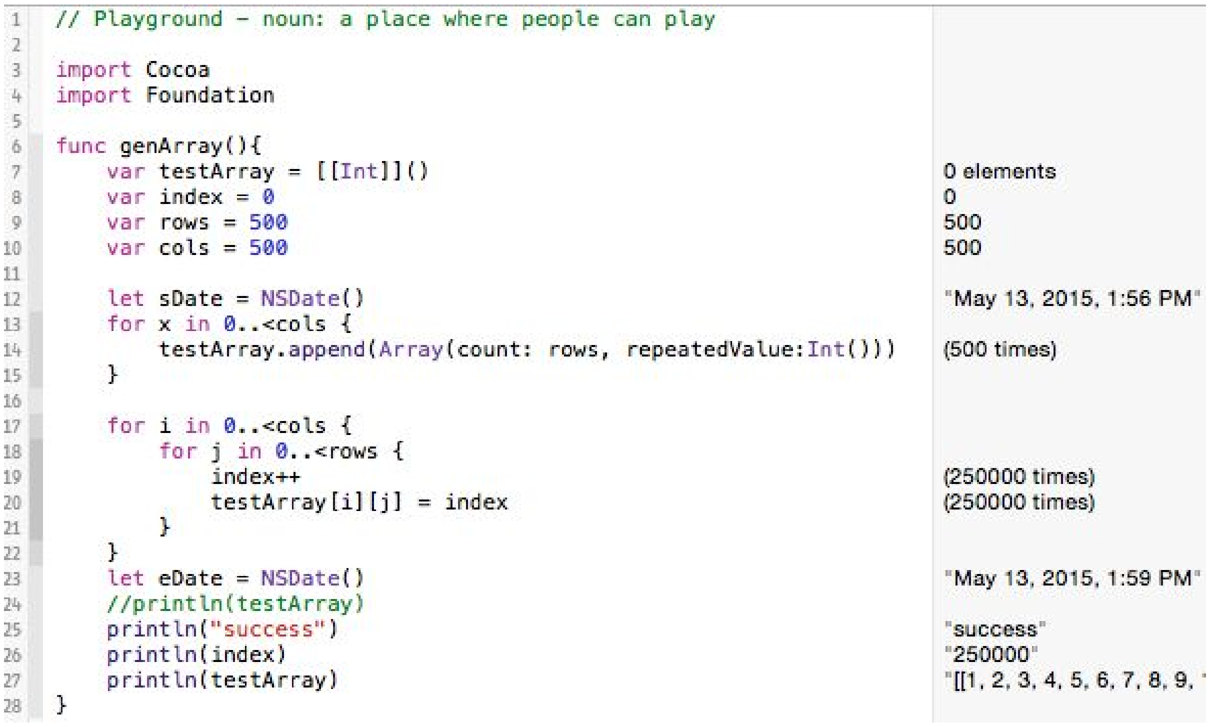
Figure 5: Python Printout Values



```
>>>
The Starting CPU Time is: 2015-05-13 12:42:44.789774
Generated the array in 0:00:05.493958
Found 3511200 in the 2d array
Found the value in the array in 0:00:00.016589
The Final CPU Time is: 2015-05-13 12:42:50.304040
The Overall Time Elapsed is: 0:00:05.514266
>>>
```

Figure 6: Swift Two Dimensional Arrray

Philip Winchester
Independent Study
Professor James Lee



```
// Playground - noun: a place where people can play

import Cocoa
import Foundation

func genArray(){
    var testArray = [[Int]]()                                              0 elements
    var index = 0                                                          0
    var rows = 500                                                         500
    var cols = 500                                                         500

    let sDate = NSDate()                                                   "May 13, 2015, 1:56 PM"
    for x in 0..<cols {
        testArray.append(Array(count: rows, repeatedValue:Int()))         (500 times)
    }

    for i in 0..<cols {
        for j in 0..<rows {
            index++                                                        (250000 times)
            testArray[i][j] = index                                        (250000 times)
        }
    }
    let eDate = NSDate()                                                   "May 13, 2015, 1:59 PM"
    //println(testArray)
    println("success")                                                     "success"
    println(index)                                                         "250000"
    println(testArray)                                                     "[[1, 2, 3, 4, 5, 6, 7, 8, 9,
}
```

Swift is a great language for its readability and writability but is definitely an Apple language designed to work specifically with Mac products. Swift is a great language for it’s job and Xcode is a intuitive IDE to assist with app development. However, Xcode is still being updated and this means that the current version (6.1.1) has some bugs and the promised intuitive debugging is not there yet. Currently the Xcode debugger actually has incredibly vague errors that takes thorough knowledge of your code and how Swift works. Xcode does have a way to line by line go through your code by inserting breakpoints, which is helpful and is the most helpful debugging tool. Overall Swift with playgrounds is a great way to learn the language because of it’s real-time compiling and real-time error-checking, but only Mac users will be able to utilize these tools. App Development is done through an intuitive UI that allows drag-and-drop connections and

Philip Winchester
Independent Study
Professor James Lee

almost seamless code interpretation. Swift is a great product but it is definitely an new language that is using a developing IDE, which has specific requirements to be used to the best of its abilities.

Swift as an iOS Tool:

The design decisions of Swift are all focused on its power an iOS Tool. In this aspect, Swift wouldn't be anything without XCode. They work together well and improve the process of designing an iOS application. Swift is a mostly finished language, but XCode is currently being updated and so is prone to crashes. Twice during my time working on my project XCode fully crashed my whole computer and many more times just crashed as an application. However, the potential for a strong tool is definitely there. Xcode's auto-fill feature means that remembering all the different NSObjects that are inevitably part of iOS developing, is not necessary. You need to be familiar with what each object is, but XCode will help you fill out the name, remind you how to initialize that NSObject, and help you remember what functions each NSObject has. None of this groundbreaking, but is a necessary tool to save time for developers.

Another strong tool in XCode, is the use of drag and drop in placing UI objects in the storyboard. This means that creating buttons, labels, and any other UI element in your apps storyboard is just a drag-drop-drag-link procedure. Meaning you can drag it in from the list of UI Objects, drop it in your page, then drag from that UI element to your code and then link those two things together. So XCode creates the necessary code that will work behind the scenes in your app. You can then change those features however you want, but Swift handles all the communication between your code and your app. This means that a developer focuses on how the app functions and how it looks, instead of managing how it operates.

The final tool that is a major aspect of iOS development is Apple Human Interface Guidelines (HIG). Apple HIG is the system, style, and best practices that Apple has determined for iOS development. This is outlined on the Apple Developer website but mostly is a persistent set of restrictions on what you can do. These restrictions are almost entirely positive, and are designed to improve the user experience of your app. However, there are some downsides. The five basic style of apps that you can use for your app, each have a specific person. I tried to use the “Tab Bar System” to allow my users to switch between different aspects of the app, however I needed to store some global information or pass that information between the tabs. However, after several weeks of trying different ways of accomplishing this, and the lack of resources for me to find answers, I came to the conclusion that the Tab Bar System is not designed for that purpose. The Tab Bar System is designed to be used with an external server, where you can query the information you need, not store all this information locally. Therefore, my simple local storage idea was not possible. I redesigned my app and got everything working, but with how new Swift and XCode are I had difficulty getting help and finishing my idea. Despite my personal story, the idea of firm HIG is something that is very beneficial to iOS development. I just wish that the information on it was more clear.

In conclusion, Swift and XCode form a formidable tool to create a myriad of apps. Learning Swift as a language is simple, everything is very high level and so there is not much complicated syntax that you need to learn, and playgrounds form a great tool for testing ideas and creating simple programs. iOS development with Swift and XCode is very user friendly and allows beginners to feel accomplished, but also means that professionals can create some impressive apps as well. For Swift to become a strong tool, it just needs more time. Time for updates in XCode so that it doesn’t crash as frequently. Time so that more online resources exist, so that new developers can learn quickly. Time so that Objective-C isn’t as prevalent, so that new developers won’t have to sift through Objective-C tools to find what they need with Swift. I look forward to continuing to work with Swift in the future and I hope to be able to revisit this report in a few years and update it after a few years of experience.