INFO1111: Computing 1A Professionalism

2023 Semester 1

Self-Learning Report

Submission number: 1

Github link: Click here to visit my project

Student name	Oscar Moore
Student ID	510549799
Topic	*Swift*
Levels already achieved	??
Levels in this report	A and B

Contents

1. Level A: Initial Understanding

1.1. Level A Demonstration

-Install and prepare an environment that supports swift
-Program and run a simple script
-Create a new model view in a separate script

1.2. Learning Approach

My approach to learning Swift mimics closely my approach to learning all programming languages. The first thing I did was first identify why I wanted to learn Swift. Prior to starting to study advanced computing at university, I had a clear goal of developing my own app. We live in a world that is heavily dominated by the apple ecosystem, hence, the logical next step was to choose a programming language that would work seamlessly with Apple's frameworks and tools.

The next step of my learning approach was understanding the basics of Swift. Understanding the basics of the language also included researching different environments that would support my project. Given that I was designing an app, Xcode would be the most appropriate platform to use, a necessity in the world of IOS app development.

In order to further research the language itself, I heavily relied on the magic of Chatgpt. The system's ability to scrape important information quickly and concisely, made the learning process seamless.

1.3. Challenges and Difficulties

Given the intuitive nature of SWIFT, I rarely ran into any technical difficulties (Other than a few thousand syntax errors). Having programmed in java and python before, Swift did not present a massive learning curve. With this being said, I ran into numerous challenges with regard to setting up the environment. Using the Xcode environment differed greatly from any IDE I had used before.

My largest challenge was the lack of resources or materials available for learning in swift. While there are many resources available for Swift, they are less abundant than resources for more established languages like Java and Python. When I started programming my project, I ran into a bug that was unique to the new IOS 16 system, which was rarely covered on the apple developer website due to the novelty of the new apple software. This took the progress of my project to an abrupt halt until an anonymous respondee on the apple developer website answered my thread question.

1.4. Learning Sources

Learning Source - What source did you use? (Note: Include source details such as links to websites, videos etc.). Contribution to Learning - How did the source contribute to your learning (i.e. what did you use the source for)?

Learning Source - What source did you use? (Note: Include source details such as links to websites, videos etc.).	Contribution to Learning - How did the source contribute to your learning (i.e. what did you use the source for)?
Swift youtube video: click here	Extremely helpful with regard to styling my swift project, also had good tips with how to organise class hierarchy.
Chatgpt	Helpful with debugging and solving syntax errors. Also gave me key functions that could be used based on use case prompts.
Apple developer	Very helpful for debugging code with errors that were not syntax.
Swift documentation: click here	Website used to access the swift documentation

1.5. Application artifacts

My initial product outlay was to create a simple landing page for the user in Swift. I added the Zeplin package to my project to further aid the design process. Zeplin aided me to find the correct button and colour functions that I needed for my design.

After a few hours of rigorous trial and error, I programmed the following:

I use a very simple view that contains a title, some text and a text field input. I used a Vstack that contains the main content of the view to store my information vertically. Inside of the Vstack, I have a horizontal stack which contains the text views with different colours and fonts. They are positioned next to each other using the "spacing" parameter of the Hstack and offset using the "offset" modifier. When the user presses "return" on the keyboard the "onCommit' closure is called, which calls a function called "login sequence" with the name of the variable as a parameter. The function will set the "ShouldShowNextView" to true and change the view to a new one (Which I havn't programmed yet).

Please proceed to the next page...

(a) preview of landing page

```
NavigationView{
    VStack {
        //MUST DOWNLOAD FONT ASSETS
        HStack(spacing: -45){
            Text("the")
                .foregroundColor(Color(red: 0/255.0, green: 0/255.0, blue: 0/255.0, opacity: 1))
                .frame(width: 122, height: 32)
                .font(Font.custom("Inter-Bold", size: 24))
            Text("Bubble")
                .foregroundColor(color)
                .font(Font.custom("Inter-Bold", size: 24))
        }
        .offset(x:-20, y:-250)

        VStack(alignment: .leading,spacing: 0){
            Text("Welcome, ")
                .foregroundColor(Color(red: 9/255.0, green: 10/255.0, blue: 10/255.0, opacity: 1))
                .font(Font.custom("Inter-Bold", size: 30))

            HStack(spacing: -45){
                Text("my nickname is")
                    .foregroundColor(Color(red: 0/255.0, green: 0/255.0, blue: 0/255.0, opacity: 1))
                    .font(Font.custom("Inter-Bold", size: 28))
                TextField(
                    "",
                    text: $name,
                    onCommit:{
                        loginSequence(var: name)
                        shouldShowNextView = true
                    })
                .offset(x:55)
                .foregroundColor(color)
                .font(Font.custom("Inter-Bold", size: 28))

            }
```

(b) Sinpet of code for landing page

Figure 1: The brunt of the information on the landing screen was included in a VStack as you can see above. This organized all my information vertically down the screen. In terms of the logic of the above code, there is very little. It is mostly formatting code so that it looks visually pleasing.

2. Level B: Basic Application

Whilst level A is about doing something simple with the topic to just show that you have started to be able to use the tool or technology, level B is about doing something practical that might actually be useful.

2.1. Level B Demonstration

In order to make the project described above actually useful, I would have to draw up an appropriate use case. I decided to make a messaging app incorporate the location of the user

-The app would consider the location of the user
-I would have pre-defined group chats that could only
be accessed when the user was physically within the pre-defined radius.

With the above requirements, the app would qualify as something that I would find useful to communicate with friends in my vicinity.

2.2. Application artifacts

The first step in providing the user with unique group chats based on location was by finding a way to pass information from a file into the Swift program. I decided to pass the parameters of the group chat (which I called bubbles in my app) using a JSON file, which my swift program read. The following screenshots include the JSON file with the necessary parameters for the 'bubble' and the config reader program I made to process this information.
Please proceed to following page...

```
{
    "bubbles":[
        {
            "id": 1,
            "name": "St johns",
            "Range": 100,
            "Private": false
        },
        {
            "id": 2,
            "name": "Sydney Uni",
            "Range": 500,
            "Private": false
        },
        {
            "id": 2,
            "name": "The Grose",
            "Range": 300,
            "Private": false
        }
        ]
}
```

(a) json example

```
import SwiftUI

struct ResponseData: Decodable {
    var bubbles: [BubbleObject]
}
struct BubbleObject: Identifiable, Decodable {
        var id: Int
        var name: String
        var Range: Int
        var Private: Bool
}

func loadJson(filename fileName: String) -> [BubbleObject]? {
    if let url = Bundle.main.url(forResource: fileName, withExtension: "json") {
        do {
            let data = try Data(contentsOf: url)
            let decoder = JSONDecoder()
            let jsonData = try decoder.decode(ResponseData.self, from: data)
            return jsonData.bubbles
        } catch {
            print("error:\(error)")
        }
    }
    return nil
}
```

(b) snipet of code used to read JSON

Figure 2: The way this works: the Swift file finds the JSON file in its local repository and passes it through a LoadJson() function. This function returns the bubble object in the correct format as defined by the structure in the code.

** What is my code doing and how it uses: variables, values, chaining classes, objects, namespaces? The code above is creating an object that can read a JSON file. I do this so that I can retrieve information from a file and implement it in using my swift code. I make a bubble object class that will act as the group chat. The bubble object has variables: id (for bubble id), name (name of group chat) , range (range of bubble) and a boolean (to state whether the group chat is private or not).
I chain this class with the ResponseData class. The role of the response data class is that it creates an array of bubble objects, so that the user can have access to multiple group chats at once.
The namespace used here is the swiftUI module. The 'import' key word at the top of the code is used to bring the code defined in SwiftUi into my file. The library allows me to output previews of my code.**

3. Level C: Deeper Understanding

Level C focuses on showing that you have actually understood the tool or technology at a relatively advanced level. You will need to compare it to alternatives, identifying key strengths and weaknesses, and the areas where this tool is most effective.

3.1. Strengths

What are the key strengths of the item you have learnt? (50-100 words)

I have learned many key strengths in Swift since undergoing my self-learning project.

I've learned about the simplicity of Swift. I've realised that Swift is an extremely intuitive and easy-to-learn language. This allows me to write clean, concise and readable code. This has also made it easier to understand and shortened the learning curve.

Its ability to be used for app development is also a large strength. Swift is specifically designed for app development, which means that it has been easy to find help and support online. Swift allows me to make powerful, high-performance apps that run seamlessly on Apple devices.

-Finally, the last strength I have learned is Swift's speed. Swift has built-in optimization features that make the language incredibly fast. It uses modern-day compiler technology to generate highly optimized code, making it faster than other programming languages.

3.2. Weaknesses

What are the key weaknesses of the item you have learnt? (50-100 words)

One of the weaknesses of using Swift is it's capability. Being a relatively new programming language, it is not fully backwards compatible with Obejctive-C, which can make it quite hard to integrate with legacy codebases.

It has limited support outside of the Apple ecosystem. Even though it is an extremely popular language for IOS and macOS development, it is not really used outside of the Apple ecosystem. Hence, it can limit the job opportunities available for a developer who specializes in swift.

Lastly, another weakness is that some tooling and libraries might not be as mature or feature-rich as those in other programming languages.

3.3. Usefulness

Describe one scenario under which you believe the topic you have learnt could be useful. (50-100 words)

A scenario under which I believe the topic I have selected could be useful is mobile development. Swift is a great language for mobile app development due to a few reasons. Firstly, it is the primary language for developing native IOS apps. Native apps have several advantages over web apps, these include better performance, access to native device features and better user experience. In addition to native app development, swift includes several features that help optimize app performance. These include automatic memory management which helps to reduce memory leaks and improve overall app performance. Swift also has built-in performance profiling tools that make it easier to optimise.

3.4. Key Question 1

Note: This question is in the table in the 'Self Learning: List of Topics' page on Canvas. (50-100 words)

There are several circumstances where Swift might be preferred over Python. Both programming languages have different strengths and weaknesses that make them better suited for different situations. The circumstances include the following:

Performance critical applications: Swift is generally faster since it is a compiled language. Compiled languages tend to be faster than interpreted languages like Python. In the circumstance that we need to write a performance-critical application such as a video game, Swift would be a better choice than Python.

Another circumstance where Swift would be preferred is if you need to develop any time of native IOS, macOS, watchOS, or TvOS applications. Since Swift is the primary language for developing native ios applications, it is preferred over other programming languages such as Python.

3.5. Key Question 2

Note: This question is in the table in the 'Self Learning: List of Topics' page on Canvas. (50-100 words)

Scenarios where using a closure might be more appropriate than a named function include:

If you need to write a simple function that will only be used once. Since it is only used only, we don't have to name it and we can use a closure to keep our code concise. They can be useful when parsing functions as arguments for other functions.

Closures are more commonly used for sorting and filtering operations. This is because they are flexible and can be used to define custom sorting and filtering criteria for a variety of data types. Hence closure is more appropriate than named function in this case.

4. Level D: Evolution of skills

4.1. Level D Demonstration

This is a short description of the application that you have developed. (50-100 words).
IMPORTANT: *You might wish to submit this as part of an earlier submission in order to obtain feedback as to whether this is likely to be acceptable for level D.*
I have created a very basic messaging app that creates pre-defined group chats based on the information in a JSON. The JSON has information about the pre-defined group chats including the radius that the group chat covers. Hence, my end product would hide group chats from users that are not in the radius

4.2. Application artifacts

Include here a description of what you actually created (what does it do? How does it work? How did you create it?). Include any code or other related artefacts that you created (these should also be included in your github repository).

If you do include screengrabs to show what you have done then these should be annotated to explain what it is showing and what the application does.

My initial product proposal for this project was to create a messaging platform at a very basic level. The issue with this initial outlay was that in order to have users communicate through my app, I would have to integrate additional software tools such as Firebase (to store user information in a database). I deduced that it would be impossible to complete the project without collaboration with other tools. I will only cover the swift components of my project in this section of the report, however, I am aware that contributions from other languages/applications would be needed to complete the entirety of the project.

The challenging part above was managing the login sequence of the user. When the user writes in their username and presses the 'enter' key, the loginsequence() function is called.

```
private func loginSequence(var nickname: String){
    let url = URL(string: "http://localhost:3000/login/hsis")!
    Auth.auth().signInAnonymously { authResult, error in
        guard let result = authResult, error == nil else {
            print("Failed to log in")
            return
        }
        let user = result.user
        print("Logged in user")
    }
```

Figure 3: Function code

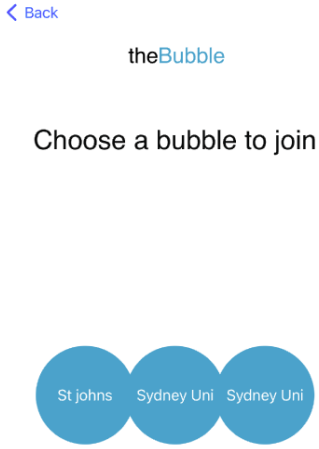
This function basically registers the user into the database and authorises it to move to the next page. This is the first place that my program interacts with firebase, the user is given a unique identifier that is stored on the database. I decided to use the anonymous user authentication function that firebase provides since I wanted to avoid a long and tedious user registration process (The create account button on the display above was used in case the user wanted to save message history). If the login sequence is successful, the user is directed to the next swift page.

Once you were navigated away from the landing page, I had to create the interface for the group chats. Since I called my group chat bubbles, I decided to use physical bubbles to represent the group chats in my app.

Once again, I stored all the contents of the page in a VStack so that all the contents would be displayed vertically. Inside the Vstack, I use Hstack which contains two text views with different colours and fonts. The first text view has an "onAppear" closure that loads the data from the JSON file (Explained in level B) into the "bubble variable" which I explain later.

There is an Hstack that contains a ForEach loop, this loops through the "bubbles" array and creates instances of BubbleCircles view for each item. The objects take parameters: name and size. The colours, fonts, sizes and positions of the views are customized using the "foreground color", "font", "frame" and "offset" modifiers.

The following code includes the logic used in order to output the bubbles onto the screen. I have also included a screenshot of what the preview of the app looked like for the respective page.



(a) preview of page

```
var body: some View {
    VStack{
        //MUST DOWNLOAD FONT ASSETS
        HStack(spacing: -45){
            Text("the")
                .foregroundColor(Color(red: 0/255.0, green: 0/255.0, blue: 0/255.0, opacity: 1))
                .frame(width: 122, height: 32)
                .font(Font.custom("Inter-Bold", size: 24))
                .onAppear{
                    bubbles = loadJson(filename: "bubble_data") ?? []
                }
            Text("Bubble")
                .foregroundColor(color)
                .font(Font.custom("Inter-Bold", size: 24))
        }
        .offset(x:-20, y:-250)

        Text("Choose a bubble to join")
            .foregroundColor(Color(red: 9/255.0, green: 10/255.0, blue: 10/255.0, opacity: 1))
            .font(Font.custom("Inter-Bold", size: 30))
            .offset(x:0, y:-200)
              .onAppear{
                  Render(BubbleList: bubbles)
              }
        HStack(spacing: -10){
              let render = Render(NumberOfCircles: CGFloat(bubbles.count))
            ForEach(bubbles) { bubble in
                BubbleCircle(name: bubble.name, size: 110)
            }
        }
```

(b) Sinpet of code for page

Figure 4: The way this works: the Swift file finds the JSON file in its local repository and passes it through a LoadJson() function. This function returns the bubble object in the correct format as defined by the structure in the code. The BubbleCircle function is called and returns the bubble circle view that you can see on the left hand side. It takes the name and size parameters and outputs the circles accordingly

This is the code for the bubble class that takes in the parameters of the bubble and outputs a bubble object. I use the in-built Circle() function to mimic the look of a bubble. I put all the contents of the bubble inside of a Zstack. The bubble has the following parameters: color, name, and size. The color of the bubble stays constant since I want all the groupchats to look the same.

```
struct BubbleCircle: View {
    let color = Color(red: 65/255.0, green: 169/255.0, blue: 202/255.0, opacity: 1)
    let name: String
    let size: CGFloat

    var body: some View {
        ZStack {
            Circle()
                .foregroundColor(color)
            Text(name)
                .foregroundColor(.white)
        }
        .frame(width: size, height: size)
    }
}
```

Figure 5: Object code

4.3. Alternative tools/technologies

2 alternative languages that can be used instead of Switft are:

Objectiv-C: Before the release of Swift, it was the primary language used for developing IOS and macOS apps. It is an object-orientated programming language and is still widely used today. Objective-C is compatible with Swift, many developers still use the language to maintain or update older apps that were previously written in Objective-C.One of the advantages of Objective-C is it's dynamic nature and allowing developers to add or change functionality at runtime.

JavaScript: It is also possible to develop cross-platform mobile applications with javascript. With the rise of frameworks like React Native and NativeScript, it is increasingly easy to develop IOS apps. This being said, JavaScript is still primarily used for developing web applications. The language is used by many companies including Facebook, Instagram and Airbnb and it allows developers to create mobile apps using a single codebase.

4.4. Comparative Analysis

Describe situations in which both your topic and each of the identified alternatives would be preferred over the others (100-200 words).

While Objective-C and JavaSCript are both viable alternatives to Swift, there are several scenarios where Swift might be preferred. Scenario: Developing a complex IOS app that requires a high level of performance, such as a real-time multiplayer game.
Swift is a compiled language that's optimised for modern hardware. Hence, it can run results faster compared to interpreted languages like JavaScript.
Type safety: Safety can be crucial in avoiding bugs and ensuring code quality in a complex IOS app. Swift's static typing can catch type-related errors at compile-time rather than at runtime.

There is also a situation where Objective-C would be preferred to the two other languages. Scenario: You are working on an IOS app that's been in development for several years and has a large codebase written in Obejctive-C. We would prefer Objective-C in this scenario over Siwft and JavaScript for the following reasons:
Objective-C is compatible with Swift and, furthermore, many of Apple's frameworks and APIs are still written in Objectiv-C. This means you can use Objective-C to maintain or update the app.
Legacy support. Objective-C has been used for IOS and MacOS development for many years. There is a very large community of developers who are experienced with Objective-C and hence there are good resources online. It may be easier t find developers who are familiar with Object-C to help maintain or update the app.

The situation where Javascript is preferred over Swift and OBjective-C. Scenario: You are developing a cross-platform mobile app that needs to run both on Android and on IOS. Javascript would be preferred here because:
It is compatible with cross-platform. It can be used for both web development and mobile app development using frameworks such as Ionic, React Native and PhoneGap. This means that Javascript would be preferred over other languages for cross-platform mobile apps.
Rapid prototyping. JavaScript is an interpreted language which means that code is tested quickly without the need for compilation. This can be useful for scenarios where you need to rapidly prototype and iterate on mobile apps.