iNurture Education Solutions
TOMORROW'S HERE

Subject: FUNDAMENTALS Of iOS APPLICATION DEVELOPMENT WITH SWIFT

Module Number - 2

Module Name: Swift Explorations

Syllabus

- Classes & Inheritance
- Structures
- Functions
- Swift Collection's
- Arrays
- Dictionaries
- Sets
- Xcode, installing Xcode
- interface builder
- iOS developer program
- creating apple iOS developer account
- Creating & Testing Simple iOS app in Simulator

AIM:

To understand the concept’s of Swift Programming Language, Overview of XCODE tool, iOS Developer Program, uOS Developer Account.

Objectives:

- Study and understand the iOS Developer Program
- Study and understand about Xcode Tool Overview
- Study and understand concepts of Swift Programming Language

Table of Contents:

Classes, Structures And Inheritance

What are classes and structures?

In Swift, classes and structures are very similar. If we really want to master Swift, it is very important to not only understand what makes classes and structures so similar, but to also understand what sets them apart, because they are the building blocks of our applications.

Apple describes them as follows:

Classes and structures are general-purpose, flexible constructs that become the building blocks of your program's code. You define properties and methods to add functionality to your classes and structures by using the already familiar syntax of constants, variables, and functions.

Let's begin by taking a quick look at some of the similarities between classes and structures.

Similarities between classes and structures

In Swift, classes and structures are more similar than they are in other languages, such as Objective-C. The following is a list of some of the features that classes and structures share:

Properties: These are used to store information in our classes and structures

Methods: These provide functionality for our classes and structures

Initializers: These are used when initializing instances of our classes and structures

Subscripts: These provide access to values using the subscript syntax

Extensions: These help extend both classes and structures

Now, let's take a quick look at some of the differences between classes and structures

Differences between classes and structures

While classes and structures are very similar, there are also several very important differences.

The following is a list of some of the differences between classes and structures in Swift:

Type: A structure is a value type, while a class is a reference type

Inheritance: A structure cannot inherit from other types, while a class can

Deinitializers: Structures cannot have custom deinitializers, while a class can

Creating a class or structure

We use the same syntax to define classes and structures. The only difference is that we define a class using the class keyword and a structure using the struct keyword. Let's look at the syntax that's used to create both classes and structures:

```
class MyClass {
 // MyClass definition
}
struct MyStruct {
 // MyStruct definition
}
```

In the preceding code, we define a new class named MyClass and a new structure named MyStruct. This effectively creates two new Swift types named MyClass and MyStruct. When we name a new type, we want to use the standard naming convention set by Swift, where the name is in camel case, with the first letter being uppercase. This is also known as PascalCase. Any method or property defined within the class or structure should also be named using camel case, with the first letter being uppercase.Empty classes and structures are not that useful, so let's look at how we can add properties to our classes and structures.

Inheritance

The concept of inheritance is a basic object-oriented development concept. Inheritance allows a class to be defined as having a certain set of characteristics, and then other classes can be derived from that class.

The derived class inherits all of the features of the class it is inheriting from (unless the derived class overrides those characteristics) and then usually adds additional characteristics of its own.

With inheritance, we can create what is known as a class hierarchy. In a class hierarchy, the class at the top of the hierarchy is known as the base class, and the derived classes are known as subclasses. We are not limited to only creating subclasses from a base class, we can also create subclasses from other subclasses. The class that a subclass is derived from is known as the parent or superclass. In Swift, a class can have only one parent class. This is known as single inheritance.

Note: Inheritance is one of the fundamental differences that separates classes from structures. Classes can be derived from a parent or superclass, but a Structure cannot.

Swift Explorations

Subclasses can call and access the properties, methods, and subscripts of their superclass. They can also override the properties, methods, and subscripts of their superclass. Subclasses can add property observers to properties that they inherit from a superclass so that they can be notified when the values of the properties change. Let's look at an example that illustrates how inheritance works in Swift.We will start off by defining a base class named Plant. The Plant class will have two properties: height and age. It will also have one method: growHeight(). The height property will represent the height of the plant, the age property will represent the age of the plant, and the growHeight() method will be used to increase the height of the plant. Here is how we would define the Plant class:

```
class Plant {
 var height = 0.0
 var age = 0
 func growHeight(inches: Double) {
 height += inches;
 }
}
```

Swift Explorations

Now that we have our Plant base class, let's see how we would define a subclass of it. We will name this subclass Tree. The Tree class will inherit the age and height properties of the Plant class and add one more property, named limbs. It will also inherit the growHeight() method of the Plant class and add two more methods: limbGrow() where new limbs are grown, and limbFall(), where limbs fall off the tree. Let's have a look at the following code:

```
class Tree: Plant {
 var limbs = 0
 func limbGrow() {
 self.limbs += 1
 }
 func limbFall() {
self.limbs -= 1
 }
}
```

Swift Explorations

We indicate that a class has a superclass by adding a colon and the name of the superclass to the end of the class definition. In this example, we indicated that the Tree class has a superclass named Plant.Now, let's look at how we could use the Tree class that inherited the age and heightproperties from the Plant class:

```
var tree = Tree()
tree.age = 5
tree.height = 4
tree.limbGrow()
tree.limbGrow()
```

The preceding example begins by creating an instance of the Tree class. We then set the Age and height properties to 5 and 4, respectively, and add two limbs to the tree by calling the limbGrow() method twice.We now have a base class named Plant that has a subclass named Tree. This means that the super (or parent) class of Tree is the Plant class. This also means that one of the subclasses (or child classes) of Plant is named Tree. There are, however, lots of different kinds of trees in the world. Let's create two subclasses frothe Tree class. These subclasses will be the PineTree class and the OakTree class:

Swift Explorations

```
class PineTree: Tree {
 var needles = 0
}
class OakTree: Tree {
 var leaves = 0
}
```

Functions

In Swift, a function is a self-contained block of code that performs a specific task. Functions are generally used to logically break our code into reusable named blocks. The function's name is used to call the function.

When we define a function, we can also optionally define one or more parameters (also known as arguments). Parameters are named values that are passed into the function by the code that calls it. These parameters are generally used within the function to perform the task of the function. We can also define default values for the parameters to simplify how the function is called.

Every Swift function has a type associated with it. This type is referred to as the return type and it defines the types of data returned from the function to the code that called it. If a value is not returned from a function, the return type is Void.

Let's look at how to define functions in Swift.

Defining a function

```
func functionName (parameters) -> ReturnType {
 // Body of the function
}
```

Example:

```
func displayPi() {
 print("3.1415926535")
}
```

Using a single parameter function

The syntax that's used to define a function in Swift is very flexible. This flexibility makes it easy for us to define simple C-style functions, or more complex functions, with local and external parameter names.

Let's look at some examples of how to define functions. The following example accepts one parameter and does not return any value back to the code that called it:

```
func sayHello(name: String) -> Void {
 let retString = "Hello " + name
 print(retString)
}
```

In the preceding example, we defined a function named sayHello() that accepted one parameter, named **name**. Inside the function, we printed out a greeting to the **name** of the person. Once the code within the function is executed, the function exits, and control is returned back to the code that called it.

Swift Explorations

Rather than printing out the greeting, we could return it to the code that called it by adding a return type, as follows:

```
func sayHello2(name: String) ->String {
 let str1 = "Hello " + name
 return str1
}
```

The -> string defines that the return type associated with the function is a string. This means that the function must return an instance of the String type to the code that calls it. Inside the function, we build a string constant, named str1, with the greeting message and then return it using the return statement.

Calling Methods Syntax

Object.Method

Calling a Swift function is a similar process to calling functions or methods in other languages, such as C or Java. The following example shows how to call the sayHello(name:) function, which prints the greeting message to the screen:

```
func sayHello(name: String) -> Void {
 let str1 = "Hello " + name
 print(str1)
}

self.sayHello(name:"Jon")
```

Swift Explorations

Now, let's look at how to call the sayHello2(name:) function, which returns a value back to the code that called it:

```
func sayHello2(name: String) ->String {
 let str1 = "Hello " + name
 return str1
}
var message = sayHello2(name:"Jon")
print(message)
```

In the preceding example, we called the sayHello2(name:) function and inputted the value that was returned in the message variable. If a function defines a return type as the sayHello2(name:) function does, it must return a value of that type to the code that called it. Therefore, every possible conditional path within the function must end by returning a value of the specified type. This does not mean that the code that called the function is required to retrieve the returned value. As an example, both lines in the
following snippet are valid**:**

```
sayHello2(name:"Jon")
var message = sayHello2(name:"Jon")
```

Swift Explorations

If you do not specify a variable for the return value to go into, the value is dropped. When the code is compiled, you will receive a warning if a function returns a value and you do not put it into a variable or a constant. You can avoid this warning by using an underscore, as shown in the following example:

```
_ = sayHello2(name:"Jon")
```

The underscore tells the compiler that you are aware of the return value, but you do not want to use it. Using the @discardableResult attribute when declaring a function will also silence the warning. This attribute is used as follows:

```
@discardableResult func sayHello2(name: String) ->String {
 let retString = "Hello " + name
 return retString
}
```

Let's look at how we would define multiple parameters for our functions

Swift Explorations

Using a multi-parameter function

We are not limited to just one parameter with our functions; we can also define multiple parameters.

To create a multi-parameter function, we list the parameters in the parentheses and separate the parameter definitions with commas.

Let's look at how to define multiple parameters in a function:

```
func sayHello(name: String, greeting: String) {
 print("\(greeting) \(name)")
}
```

In the preceding example, the function accepts two arguments: name and greeting. We then print a greeting to the console using both parameters.

Swift Explorations

Calling a multi-parameter function is a little different from calling a single-parameter function. When calling a multi-parameter function, we separate the parameters with commas. We also need to include the parameter name for all the parameters. The following example shows how to call a multi-parameter function:

```
func sayHello(name: String, greeting: String) {
 print("\(greeting) \(name)")
}
sayHello(name:"Jon", greeting:"Bonjour")
```

We do not need to supply an argument for each parameter of the function if we define default values. Let's look at how to configure default values for our parameters.

Defining a parameter's default values

We can define default values for any parameter by using the equals to operator (=) within the function definition when we declare the parameters. The following example shows how to declare a function with a parameter's default values:

```
func sayHello(name: String, greeting: String = "Bonjour") {
 print("\(greeting) \(name)")
}
```

In the function declaration, we have defined one parameter without a default value (name:String) and one parameter with a default value (greeting: String = "Bonjour"). When a parameter has a default value declared, we are able to call the function with or without setting a value for that parameter.

The following example shows how to call the sayHello() function without setting the greeting parameter, and also how to call it when you do set the greeting parameter:

```
sayHello(name:"Jon")
sayHello(name:"Jon", greeting: "Hello")
```

In the sayHello(name:"Jon") line, the function will print out the message Bonjour Jon since it uses the default value for the greeting parameter. In the sayHello(name:"Jon", greeting: "Hello") line, the function will print out the message Hello Jon since we have overridden the default value for the greeting parameter.

Swift Explorations

We can declare multiple parameters with default values and override only the ones we want by using the parameter names. The following example shows how we would do this by overriding one of the default values when we call it:

```
func sayHello(name: String = "Test", name2: String = "Kim", greeting:
String = "Bonjour") {
 print("\(greeting) \(name) and \(name2)")
}
sayHello(name:"Jon",greeting: "Hello")
```

In the preceding example, we declared a function with three parameters, each with a default value. We then called the function, leaving the name2 parameter with its default value, while overriding the default values for the remaining two parameters.

The preceding example will print out the message **Hello Jon and Kim**.

Swift Explorations

Omitting argument labels

All of the functions in this chapter have used labels when passing arguments into the functions. If we do not want to use a label, we can omit it by using an underscore. The following example illustrates this:

```
func sayHello(_ name: String, greeting: String) {
 print("\(greeting) \(name)")
}
```

Notice the underscore prior to the name label in the parameter list. This indicates that the name label should not be used when calling this function. Now, we are able to call this function without using the name label:

```
sayHello("Jon", greeting: "Hi")
```

This call would print out **Hi Jon**.

Collections

Swift collection types

A collection groups multiple items into a single unit. Swift provides three native collection types. These collection types are arrays, sets, and dictionaries. Arrays store data in an ordered collection, sets are unordered collections of unique values, and dictionaries are unordered collections of key-value pairs. In an array, we access the data by the location or index in the array, whereas in a set we usually iterate through the collection, and dictionaries are accessed using a unique key.

The data stored in a Swift collection must be of the same type. This means, as an example, that we are unable to store a string value in an array of integers. Since Swift does not allow us to mismatch data types in a collection, we can be certain of the data type when we retrieve elements from a collection. This is another feature that, on the surface, might seem like a shortcoming, but actually helps eliminate common programming mistakes.

1. Arrays

Arrays are a very common component of modern programming languages and can be found in virtually all modern programming languages. In Swift, an array is an ordered list of objects of the same type.

When an array is created, we must declare the type of data that can be stored in it by explicit type declaration or through type inference. Typically, we only explicitly declare the data type of an array when we are creating an empty array. If we initialize an array with data, the compiler uses type inference to infer the data type for the array.

Each object in an array is called an element. Each of these elements is stored in a set order and can be accessed by searching for its location (index) in the array.

Swift Explorations

Creating and initializing arrays

We can initialize an array with an array literal. An array literal is a set of values that

prepopulates the array. The following example shows how to define an immutable array of integers using the let keyword:

```
let arrayOne = [1,2,3]
```

If we need to create a mutable array, we will use the var keyword to define the array, as we did with standard variables. The following example shows how to define a mutable array:

```
var arrayTwo = [4,5,6]
```

In the preceding two examples, the compiler inferred the type of values stored in the array by looking at the type of values stored in the array literal. If we want to create an empty array, we need to explicitly declare the type of values to store in the array. There are two ways to declare null arrays in Swift.

Swift Explorations

The following examples show how to declare an empty mutable array that can be used to store integers:

```
var arrayThree = [Int]()
var arrayThree: [Int] = []
```

In the preceding examples, we created arrays with integer values, and the majority of the array examples in this chapter will also use integer values; however, we can create arrays in Swift with any type. The only rule is that, once an array is defined as containing a particular type, all the elements in the array must be of that type. The following example shows how

we can create arrays of various data types:

```
var arrayOne = [String]()
var arrayTwo = [Double]()
var arrayThree = [MyObject]()
```

Swift provides special type aliases for working with nonspecific types. These aliases are AnyObject and Any. We can use these aliases to define arrays whose elements are of different types, like this:

```
var myArray: [Any] = [1,"Two"]
```

Swift Explorations

The AnyObject aliases can represent an instance of any class type, while the Any aliases can represent an instance of the instance of any type, including function types. We should use the Any and AnyObject aliases only when there is an explicit need for this behavior. It is always better to be specific about the types of data our collections contain.

Arrays contains

```
let numbers = [4, 5, 6]
if numbers.contains(5) {
 print("There is a 5")
}
There is a 5
```

Array types

```
var myArray: [Int] = []
var myArray: Array<Int> = []
var myArray = [Int]()
```

Working with arrays

repeating

```
var myArray = [Int](repeating: 0, count: 100)
let count = myArray.count
if myArray.isEmpty { }
```

Working with arrays

Accessing or setting a specific item

```
var names = ["Anne", "Gary", "Keith"]
let firstName = names[0]
print(firstName)
Anne
names[1] = "Paul"
print(names)
["Anne", "Paul", "Keith"]
```

Working with arrays

Appending

```
var names = ["Amy"]
names.append("Joe")
names += ["Keith", "Jane"]
print(names)
["Amy", "Joe", "Keith", "Jane"]
```

Swift Explorations

Working with arrays

Inserting

```
var names = ["Amy", "Brad", "Chelsea", "Dan"]
names.insert("Bob", at: 0)
print(names)
["Bob", "Amy", "Brad", "Chelsea", "Dan"]
```

Working with arrays

Removing

```
var names = ["Amy", "Brad", "Chelsea", "Dan"]
let chelsea = names.remove(at:2)
let dan = names.removeLast()
print(names)
["Amy", "Brad"]
names.removeAll()
print(names)
[]
```

Swift Explorations

Working with arrays

Arrays within arrays

```
let array1 = [1,2,3]
let array2 = [4,5,6]
let containerArray = [array1, array2]
let firstArray = containerArray[0]
let firstElement = containerArray[0][0]
print(containerArray)
print(firstArray)
print(firstElement)
```

```
[[1, 2, 3], [4, 5, 6]]
[1, 2, 3]
1
```

2. Dictionaries

While dictionaries are not as commonly used as arrays, they have additional functionality that makes them incredibly powerful. A dictionary is a container that stores multiple key□value pairs, where all the keys are of the same type and all the values are of the same type. The key is used as a unique identifier for the value. A dictionary does not guarantee the order in which the key-value pairs are stored since we look up the values by the key rather than by the index of the value.

Dictionaries are good for storing items that map to unique identifiers, where the unique identifier should be used to retrieve the item. Countries with their abbreviations are a good example of items that can be stored in a dictionary. In the following chart, we show countries with their abbreviations as key-value pairs:

Key	**Value**
US	United States
IN	India
UK	United Kingdom

Creating and initializing dictionaries

We can initialize a dictionary using a dictionary literal, similarly to how we initialized an array with the array literal. The following example shows how to create a dictionary usingbthe key-value pairs in the preceding chart:

```
let countries = ["US":"UnitedStates","IN":"India","UK":"UnitedKingdom"]
```

The preceding code creates an immutable dictionary that contains each of the key-value pairs in the chart we saw before. Just like the array, to create a mutable dictionary we will need to use the var keyword in place of let. The following example shows how to create a mutable dictionary that contains the countries:

```
var countries = ["US":"UnitedStates","IN":"India","UK":"UnitedKingdom"]
```

Swift Explorations

In the preceding two examples, we created a dictionary where the key and value were both strings. The compiler inferred that the key and value were strings because that was the type of the keys and values used to initiate the dictionary. If we wanted to create an empty dictionary, we would need to tell the compiler what the key and value types are. The following examples create various dictionaries with different key-value types:

```
var dic1 = [String:String]()
var dic2 = [Int:String]()
var dic3 = [String:MyObject]()
var dic4: [String:String] = [:]
var dic5: [Int:String] = [:]
```

Add/remove/modify a dictionary

Adding or modifying

```
var scores = ["Richard": 500, "Luke": 400, "Cheryl": 800]
scores["Oli"] = 399
if let oldValue = scores.updateValue(100, forKey: "Richard") {
 print("Richard's old value was \(oldValue)")
}
```

```
Richard's old value was 500
```

Swift Explorations

Add/remove/modify a dictionary

Removing

```
var scores = ["Richard": 100, "Luke": 400, "Cheryl": 800]
scores["Richard"] = nil
print(scores)
if let oldValue = scores.removeValue(forKey: "Luke") {
 print("Luke's score was \(oldValue) before he stopped playing")
}
print(scores)
```

```
["Cheryl": 800, "Luke": 400]
Luke's score was 400 before he stopped playing
["Cheryl": 800
```

Swift Explorations

Accessing a dictionary

```
var scores = ["Richard": 500, "Luke": 400, "Cheryl": 800]
let players = Array(scores.keys) //["Richard", "Luke", "Cheryl"]
let points = Array(scores.values) //[500, 400, 800]
if let myScore = scores["Luke"] {
 print(myScore)
}
400
if let henrysScore = scores["Henry"] {
 print(henrysScore)
}
```

3. Sets

The set type is a generic collection that is similar to the array type. While the array type is an ordered collection that may contain duplicate items, the set type is an unordered collection where each item must be unique.

Like the key in a dictionary, the type stored in an array must conform to the Hashable protocol. This means that the type must provide a way to compute a hash value for itself.

All of Swift's basic types, such as String, Double, Int, and Bool, conform to this protocol and can be used in a set by default. Let's look at how we would use the set type.

Initializing a set

There are a couple of ways to initialize a set. Just like the array and dictionary types, Swift needs to know what type of data is going to be stored in it. This means that we must either tell Swift the type of data to store in the set or initialize it with some data so that it can infer the data type.
Just like the array and dictionary types, we use the var and let keywords to declare whether the set is mutable:

```
//Initializes an empty set of the String type
var mySet = Set<String>()

//Initializes a mutable set of the String type with initial values

var mySet = Set(["one", "two", "three"])

//Creates a immutable set of the String type.
let mySet = Set(["one", "two", "three"])
```

Inserting items into a set

We use the insert method to insert an item into a set. If we attempt to insert an item that is already in the set, the item will be ignored. Here are some examples of inserting items into a set:

```
var mySet = Set<String>() mySet.insert("One")
mySet.insert("Two")
mySet.insert("Three")
```

The insert() method returns a tuple that we can use to verify that the value was successfully added to the set. The following example shows how to check the returned value to see whether it was added successfully:

Swift Explorations

```
var mySet = Set<String>() mySet.insert("One")
mySet.insert("Two")
var results = mySet.insert("One")
if results.inserted {
 print("Success")
} else {
 print("Failed")
}
```

In this example, Failed would be printed to the console since we are attempting to add the One value to the set when it is already in the set.

Determining the number of items in a set

We can use the count property to determine the number of items in a set. Here is an example of how to use this method:

```
var mySet = Set<String>() mySet.insert("One")
mySet.insert("Two")
mySet.insert("Three")
print("\(mySet.count) items")
```

When executed, this code will print the message 3 items to the console because the set contains three items.

Checking whether a set contains an item

We can verify whether a set contains an item by using the contains() method, as shown here:

```
var mySet = Set<String>()
mySet.insert("One")
mySet.insert("Two")
mySet.insert("Three")
var contain = mySet.contains("Two")
```

In the preceding example, the contain variable is set to true because the set contains the Two string.

Swift Explorations

Iterating over a set

We can use the for-in statement to iterate over the items in a set as we did with arrays.

The following example shows how we would iterate through the items in a set:

```
for item in mySet {
 print(item)
}
```

The preceding example will print out each item in the set to the console.

Removing items in a set

We can remove a single item or all the items in a set. To remove a single item, we would use the remove() method and, to remove all the items, we would use the removeAll() method. The following example shows how to remove items from a set:

```
//The remove method will return and remove an item from a set

var item = mySet.remove("Two")

//The removeAll method will remove all items from a set

mySet.removeAll()
```

XCODE

What is Xcode?

- Xcode is the Integrated Development Environment(IDE) provided by Apple to third-party programmers such as yourself for developing iOS and OS X applications written in Objective-C and Swift.

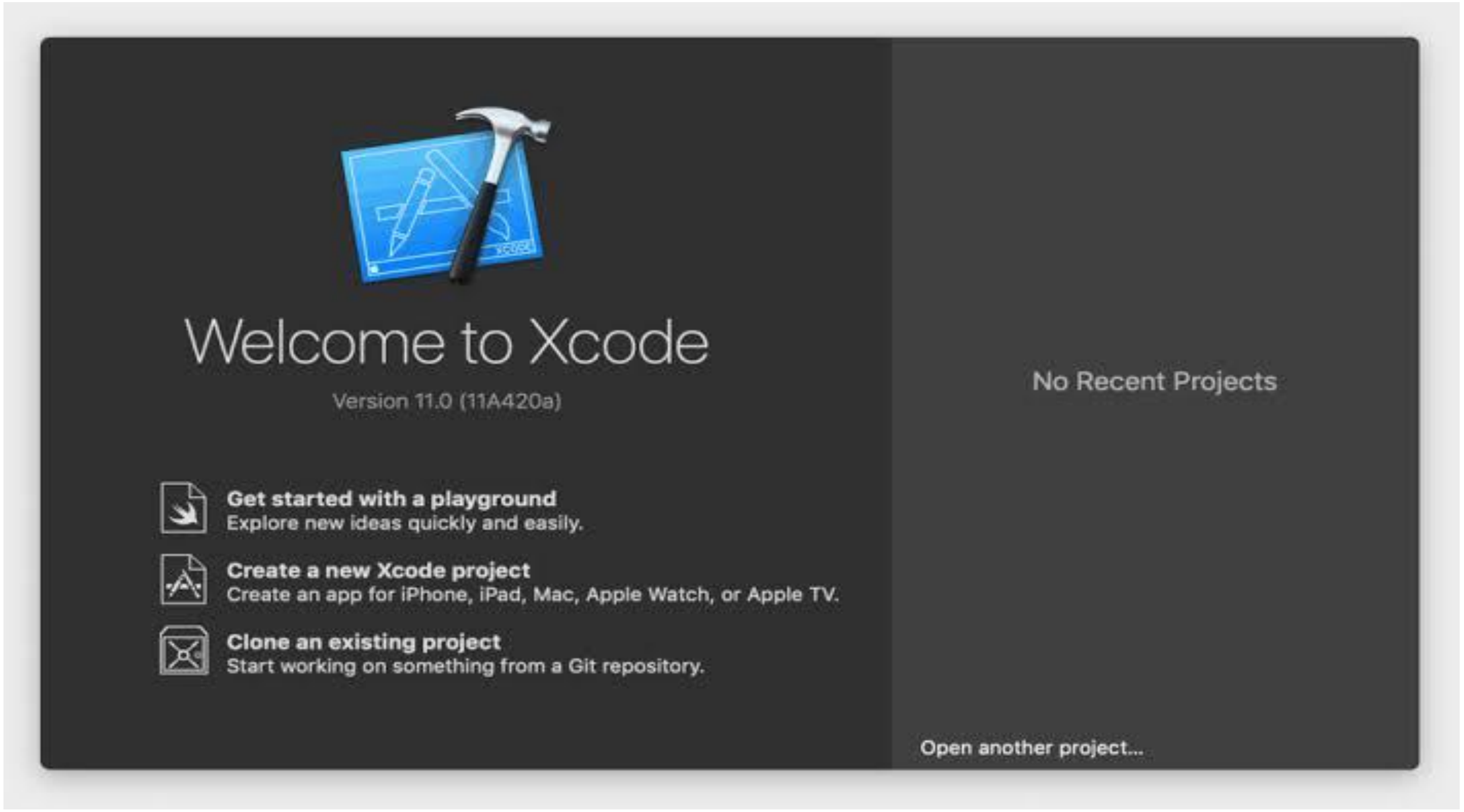
- An IDE is a program (or program suite) that incorporates tools for the various aspects of creating software. Xcode, specifically, brings together the code editor, compiler, debugger, interface builder, application bundler, and simulator (which is technically a separate program launched from within Xcode).

Swift Explorations

Xcode Templates:

1. SingleView Application
2. Game Application
3. Augmented Reality Application
4. Document Based Application
5. Master-Detail Application
6. Page-Based Application
7. Tabbed Application
8. Sticker Pack Application
9. iMessage Application

XCode Home Page



Welcome to Xcode

Version 11.0 (11A420a)

Get started with a playground
Explore new ideas quickly and easily.

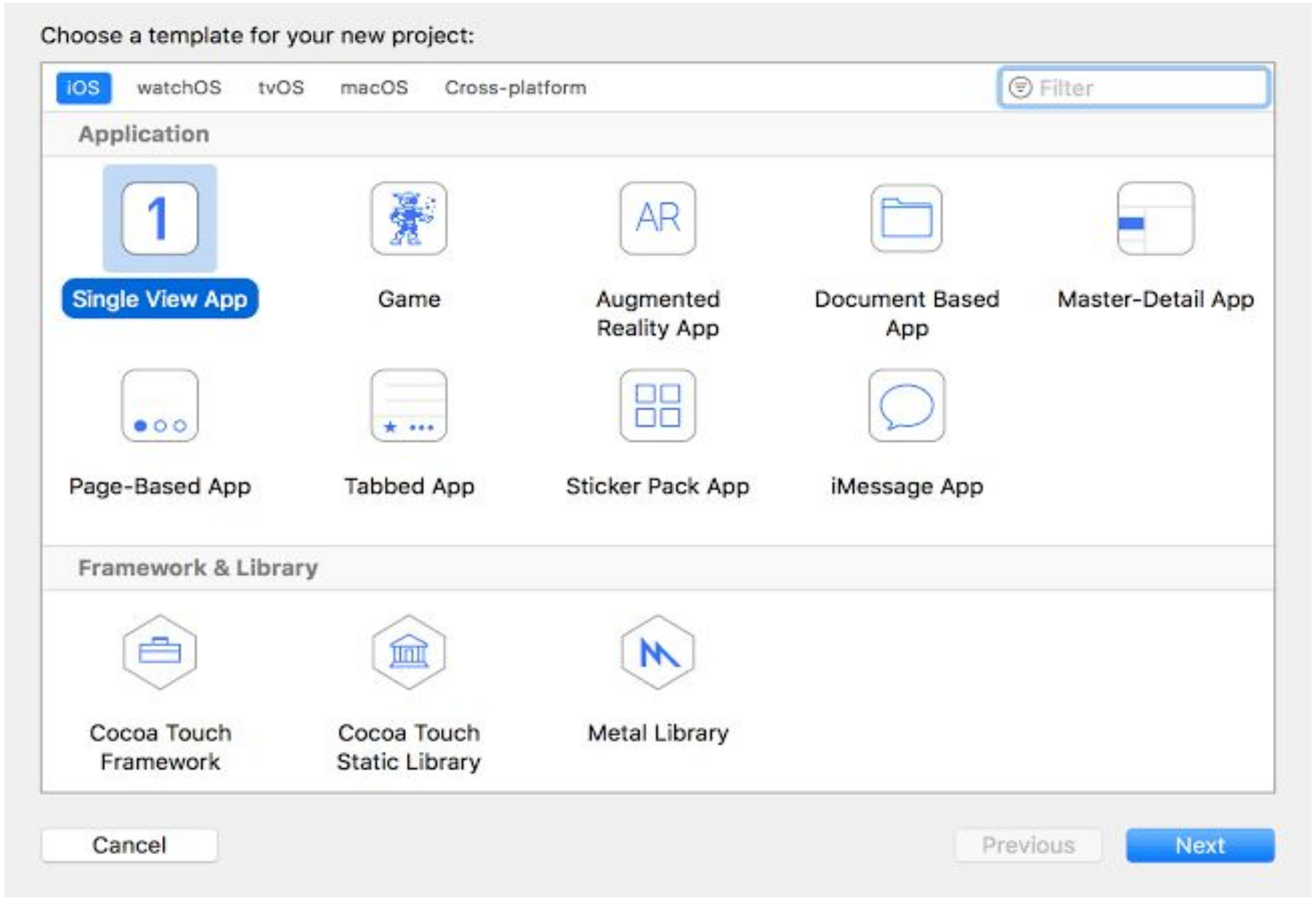
Create a new Xcode project
Create an app for iPhone, iPad, Mac, Apple Watch, or Apple TV.

Clone an existing project
Start working on something from a Git repository.

No Recent Projects

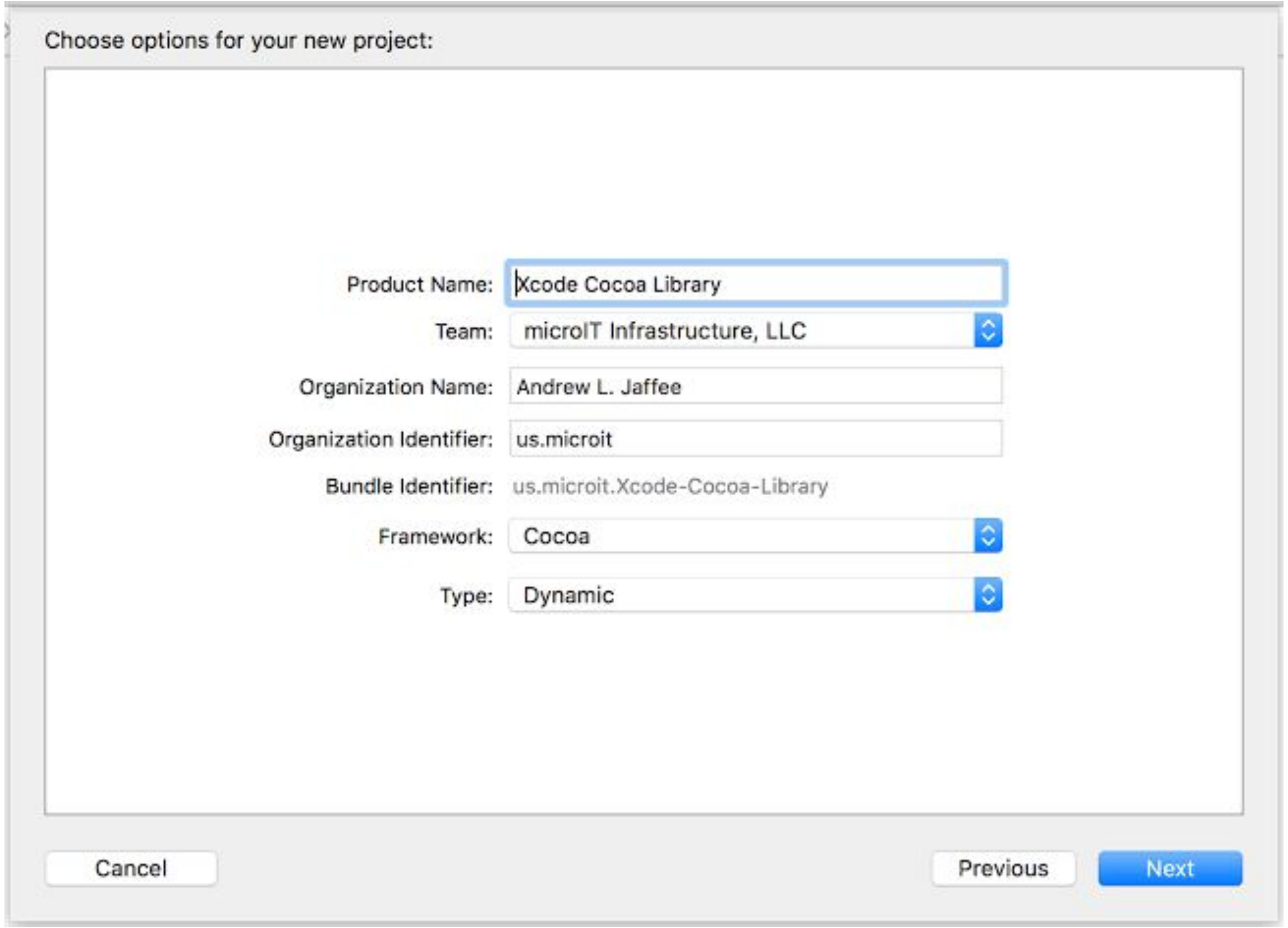
Open another project...

Swift Explorations



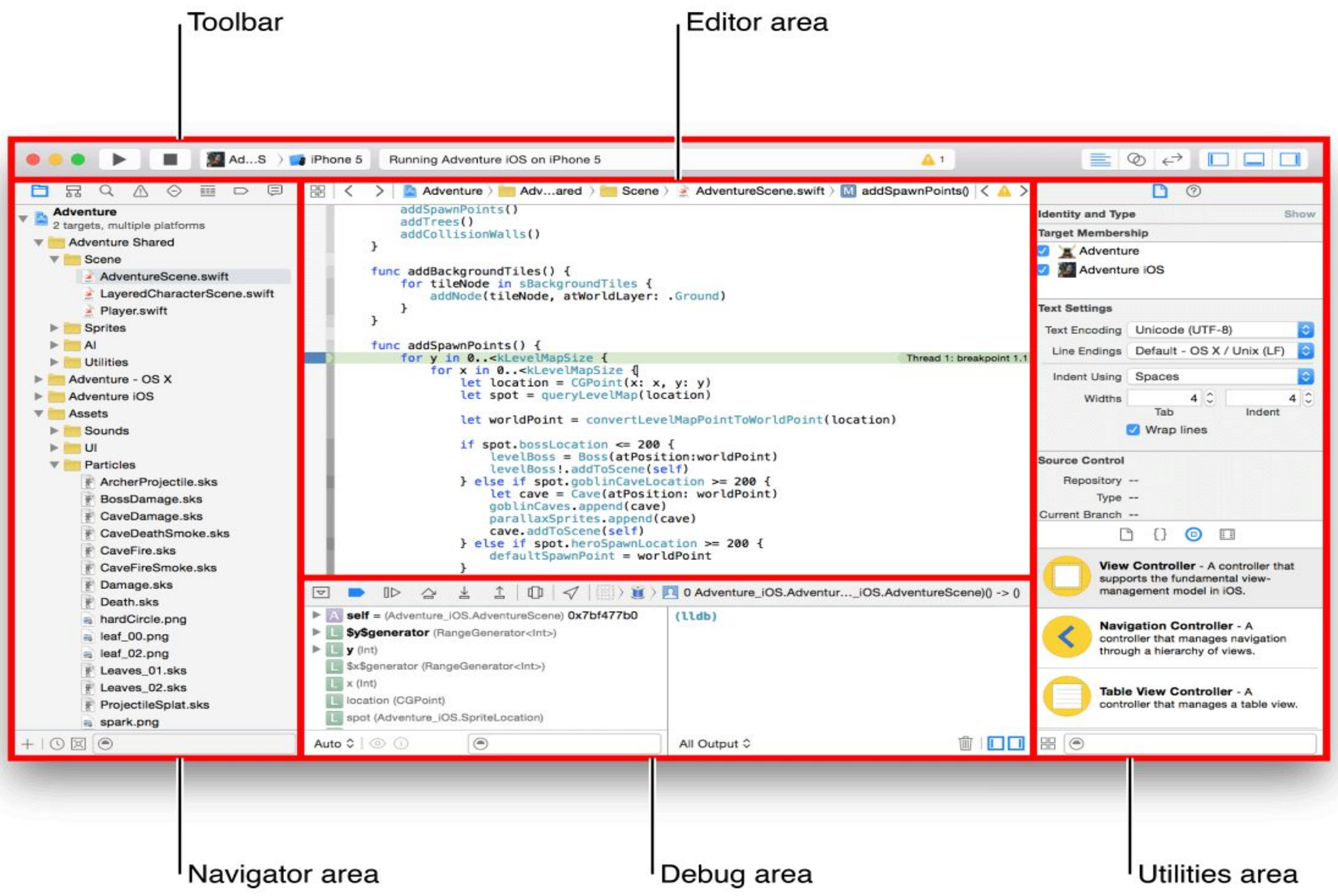
Choose a template for your new project:

Swift Explorations


Choose options for your new project:

Product Name:	Xcode Cocoa Library
Team:	microIT Infrastructure, LLC
Organization Name:	Andrew L. Jaffee
Organization Identifier:	us.microit
Bundle Identifier:	us.microit.Xcode-Cocoa-Library
Framework:	Cocoa
Type:	Dynamic

Xcode WorkSpace Windows:



```
        addSpawnPoints()
        addTrees()
        addCollisionWalls()
    }

    func addBackgroundTiles() {
        for tileNode in sBackgroundTiles {
            addNode(tileNode, atWorldLayer: .Ground)
        }
    }

    func addSpawnPoints() {
        for y in 0..<kLevelMapSize {                          Thread 1: breakpoint 1.1
            for x in 0..<kLevelMapSize {
                let location = CGPoint(x: x, y: y)
                let spot = queryLevelMap(location)

                let worldPoint = convertLevelMapPointToWorldPoint(location)

                if spot.bossLocation <= 200 {
                    levelBoss = Boss(atPosition:worldPoint)
                    levelBoss!.addToScene(self)
                } else if spot.goblinCaveLocation >= 200 {
                    let cave = Cave(atPosition: worldPoint)
                    goblinCaves.append(cave)
                    parallaxSprites.append(cave)
                    cave.addToScene(self)
                } else if spot.heroSpawnLocation >= 200 {
                    defaultSpawnPoint = worldPoint
                }
```

```
0 Adventure_iOS.Adventur..._iOS.AdventureScene)() -> ()

self = (Adventure_iOS.AdventureScene) 0x7bf477b0
$y$generator (RangeGenerator<Int>)
y (Int)
$x$generator (RangeGenerator<Int>)
x (Int)
location (CGPoint)
spot (Adventure_iOS.SpriteLocation)

(lldb)
```

Text Encoding: Unicode (UTF-8)
Line Endings: Default - OS X / Unix (LF)
Indent Using: Spaces
Widths: 4 Tab, 4 Indent
✓ Wrap lines

View Controller - A controller that supports the fundamental view-management model in iOS.

Navigation Controller - A controller that manages navigation through a hierarchy of views.

Table View Controller - A controller that manages a table view.

Swift Explorations

1. Xcode Tool Bar

The Toolbar is a small collection of project-wide buttons and labels. They are:

1. Close/Minimise/Maximise Window buttons (these are the red, yellow, and green dots standard to all OS X windows),
2. The ▶ "Run" button (⌘R) which builds and then runs the current scheme,
3. The ■ "Stop" button (⌘.) which stops the running scheme or application,

4.The "Scheme Menu",
The left half selects the current target,
The right half selects the destination—the device or simulator you wish to run on,

5. The "Activity viewer" displays information relevant to the current operation, as well as symbols for the number of warning and errors generated by the compiler,

6. The Editor configuration buttons,

- **Standard Editor** - shows the primary code editor window
- **Assistant Editor** - shows a secondary code editor window alongside the primary window (this gets crowded without a larger screen),
- **Version Editor** - shows the Xcode's built-in version control UI, since we utilise git and GitHub for version control, you will not use for the labs,

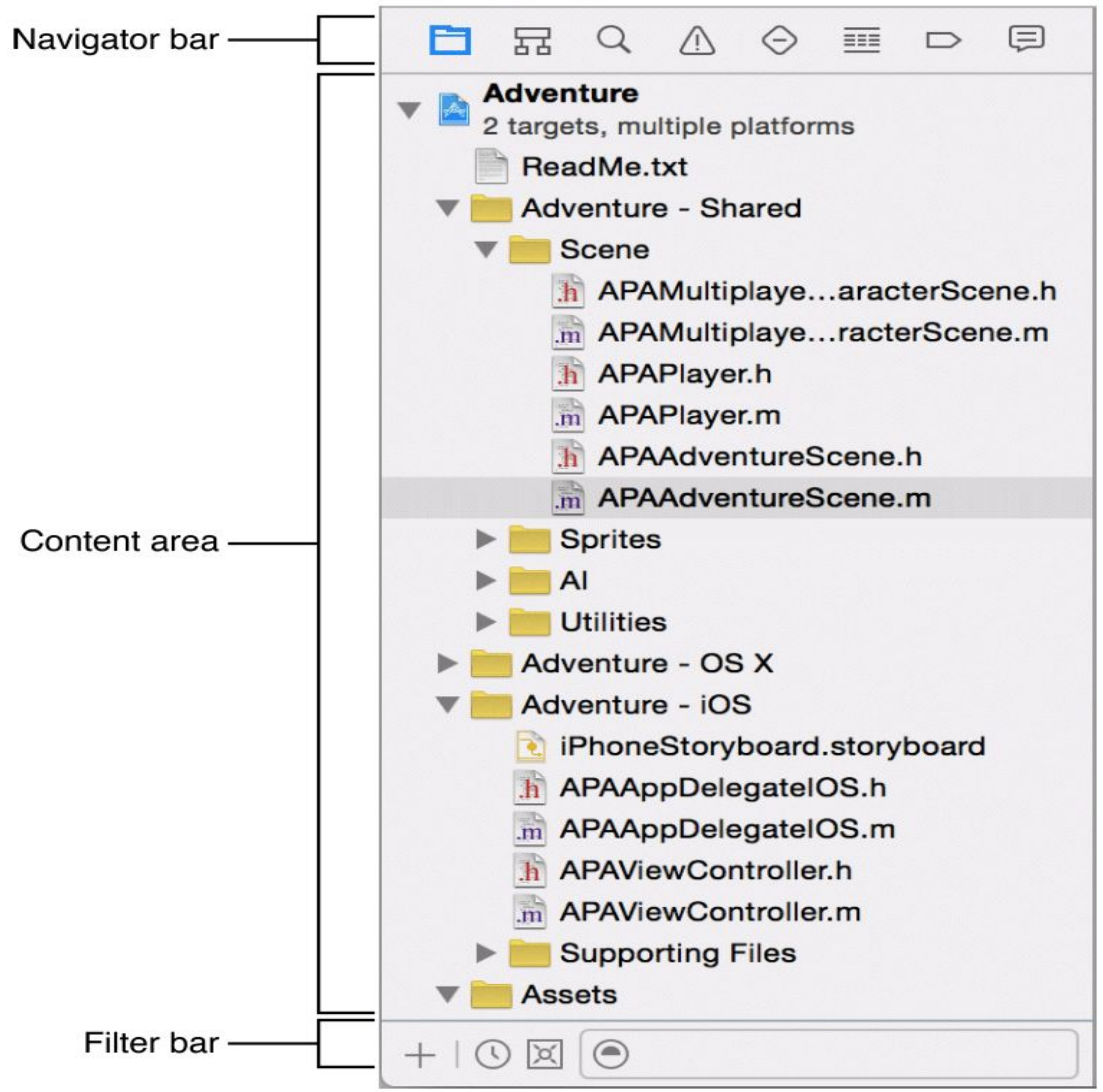
Swift Explorations

2. Navigator Area

- The appropriately-named Navigator area is actually a stack of eight different navigators which help you get around your project based on different elements.

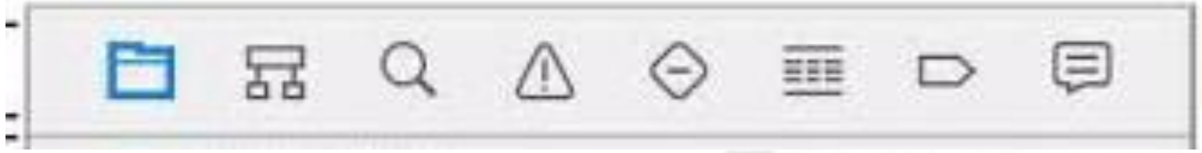
- You'll likely spend the most time with the Project Navigator which is the default drop-down file list.

Swift Explorations



Adventure
2 targets, multiple platforms
ReadMe.txt
Adventure - Shared
Scene
APAMultiplaye...aracterScene.h
APAMultiplaye...racterScene.m
APAPlayer.h
APAPlayer.m
APAAdventureScene.h
APAAdventureScene.m
Sprites
AI
Utilities
Adventure - OS X
Adventure - iOS
iPhoneStoryboard.storyboard
APAAppDelegateIOS.h
APAAppDelegateIOS.m
APAViewController.h
APAViewController.m
Supporting Files
Assets

Swift Explorations



Icon	Navigator Pane	Description
	Project	Add, delete, group, and otherwise manage files in your project, or choose a file to view or edit its contents in the editor area.
	Symbol	Browse the symbols in your project as a list or hierarchy. Buttons on the left of the filter bar let you limit the shown symbols to a combination of only classes and protocols, only symbols in your project, or only containers.
	Find	Use search options and filters to quickly find any string within your project.
	Issue	View issues such as diagnostics, warnings, and errors found when opening, analysing, and building your project.
	Test	Create, manage, run, and review unit tests.
	Debug	Examine the running threads and associated stack information at a specified point or time during program execution.
	Breakpoint	Fine-tune breakpoints by specifying characteristics such as triggering conditions.
	Report	View the history of your build, run, debug, continuous integration, and source control tasks.

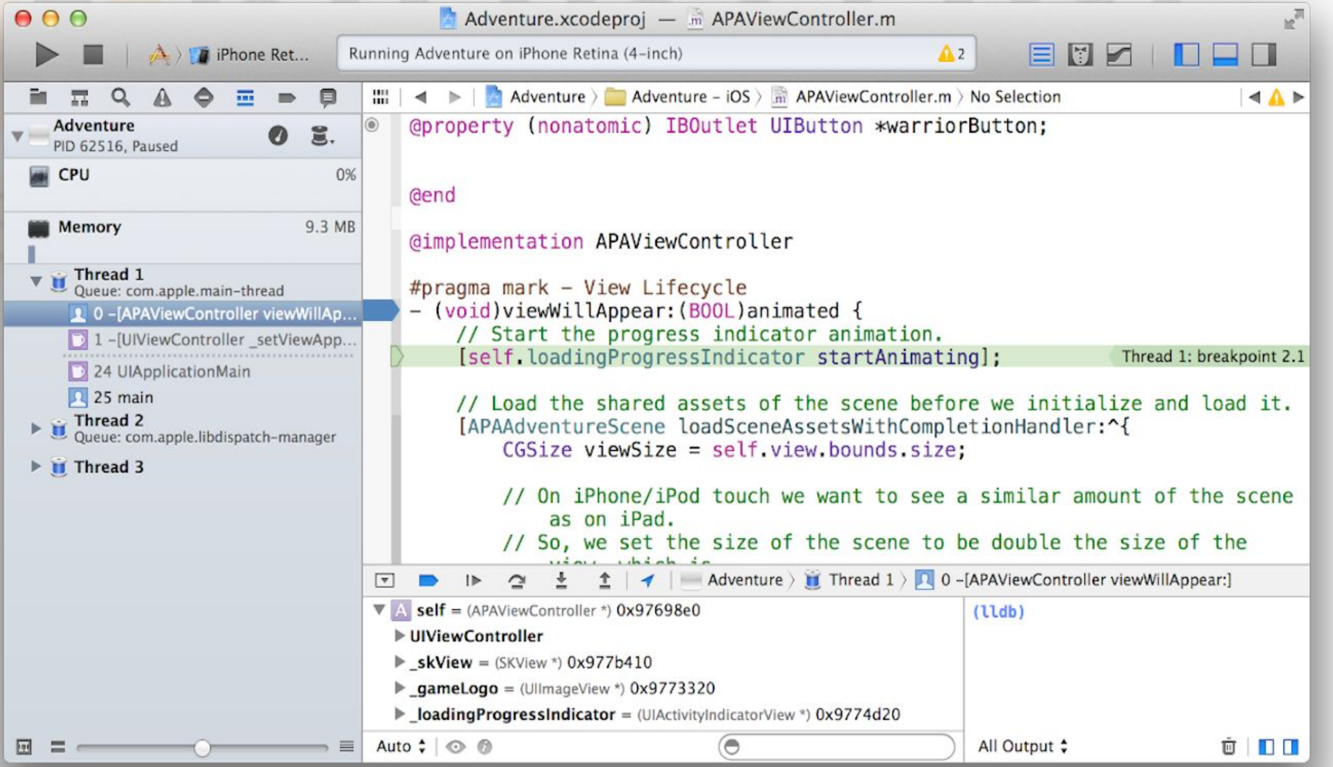
Swift Explorations

3. Editor Area

- This is the window which contains Xcode's text editor for writing code.
- The font colouring is based upon the syntax of the currently relevant programmi language.
- You can customise these colours and the font type & size of your text in Xcode's preferences under the "Fonts & Colours" tab.
- Top Tip: You can also create a presentation preset for easily switching to larger display fonts without affecting your personal font customisations.

The Gutter

- The left-most column in the Code Editor which contains the line numbers is known as the Gutter. Blue arrow-tabs present in the gutter are "breakpoint indicators". Breakpoints are a part of the debugging process that we'll teach you about later.


Adventure.xcodeproj — APAViewController.m
iPhone Ret...
Running Adventure on iPhone Retina (4-inch)
Adventure PID 62516, Paused
CPU 0%
Memory 9.3 MB
Thread 1 Queue: com.apple.main-thread
0 -[APAViewController viewWillAp...
1 -[UIViewController _setViewApp...
24 UIApplicationMain
25 main
Thread 2 Queue: com.apple.libdispatch-manager
Thread 3
Adventure › Adventure - iOS › APAViewController.m › No Selection
Adventure › Thread 1 › 0 -[APAViewController viewWillAppear:]
Auto
All Output

```
@property (nonatomic) IBOutlet UIButton *warriorButton;

@end

@implementation APAViewController

#pragma mark - View Lifecycle
- (void)viewWillAppear:(BOOL)animated {
    // Start the progress indicator animation.
    [self.loadingProgressIndicator startAnimating];          Thread 1: breakpoint 2.1

    // Load the shared assets of the scene before we initialize and load it.
    [APAAdventureScene loadSceneAssetsWithCompletionHandler:^{
        CGSize viewSize = self.view.bounds.size;

        // On iPhone/iPod touch we want to see a similar amount of the scene
           as on iPad.
        // So, we set the size of the scene to be double the size of the
```

```
▼ self = (APAViewController *) 0x97698e0
  ▶ UIViewController
  ▶ _skView = (SKView *) 0x977b410
  ▶ _gameLogo = (UIImageView *) 0x9773320
  ▶ _loadingProgressIndicator = (UIActivityIndicatorView *) 0x9774d20
```

```
(lldb)
```

Swift Explorations

- The vertical line between the line numbers and your code allows you to fold or unfold your code.

- **Code folding** is an action that collapses a block or segment of code in the editor to make it easier to read.

- This feature won't be relevant to you until you get to the more advanced labs, but you should be aware of it in case you invoke it accidentally.

- Double-clicking on this folding bar will cause the selected block of code to collapse into a pair of curly braces containing an ellipsis { ... }.

- Don't panic when you do this accidentally! Just click on the ellipsis and your code will expand back out.

4. Utilities Area

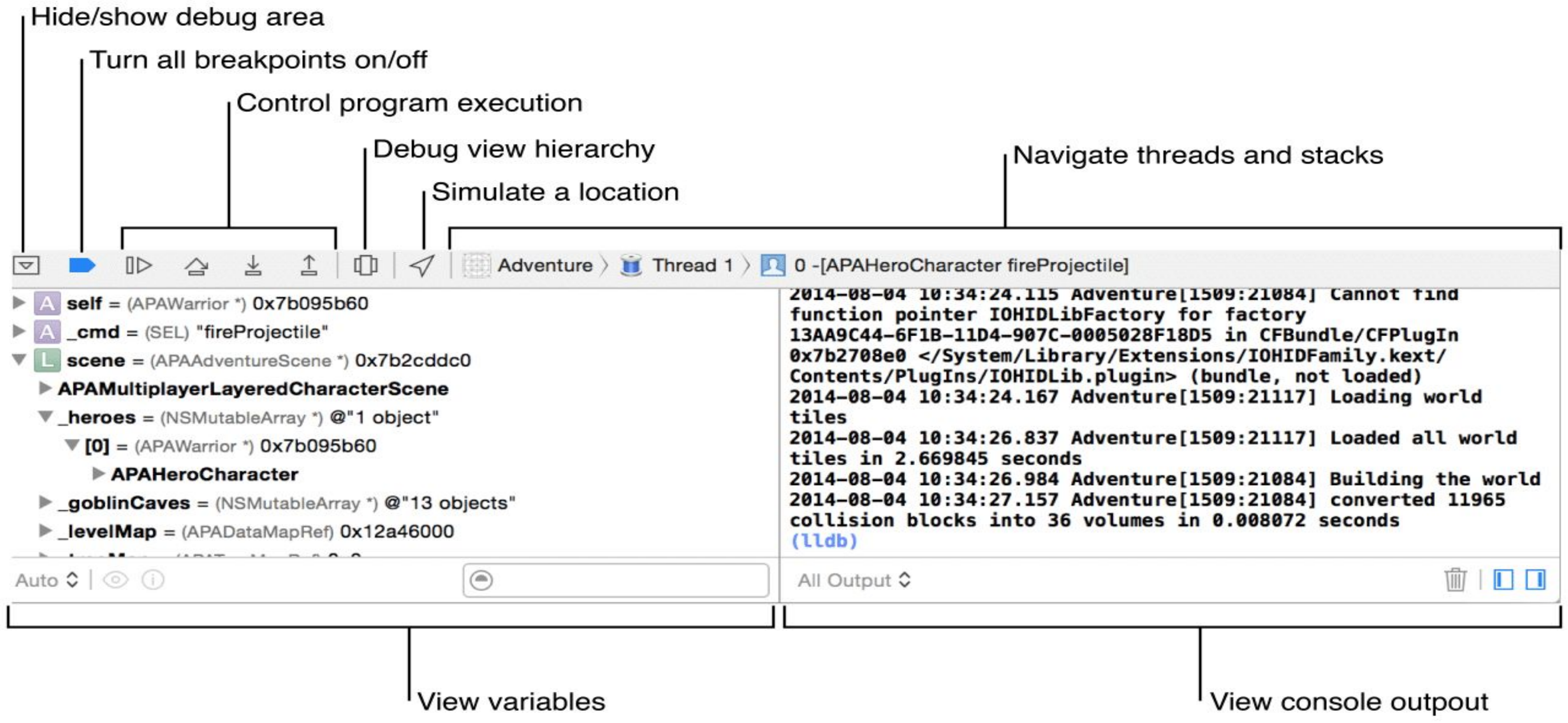
The Utilities Area really shines in Interface Builder which we're not introducing to you just yet. In the Code Editor, however, it displays only either the File Inspector or the Quick Help guide.

The File Inspector: displays a set of information relevant to the current file such as directory path and target.

The Quick Help: guide is immensely helpful. It provides a summary description of the cursor-selected class and several relevant hot links, specifically the link to Apple's Reference Guide on that class. This is often quicker than a google search and works offline, being stored locally as part of Xcode. You can access this database directly in Xcode's status bar by selecting Help —> Documentation and API Reference.

5. Debug Area

The debugger is usually tucked away while writing code, yet comes out to play almost every time the build is run. An automatic breakpoint will get triggered if your build succeeds but your program crashes. The information printed out can be a valuable source of insight toward finding the problem. In time, you'll learn how to interpret much of the information that gets dumped out by a crash.



```
self = (APAWarrior *) 0x7b095b60
_cmd = (SEL) "fireProjectile"
scene = (APAAdventureScene *) 0x7b2cddc0
  APAMultiplayerLayeredCharacterScene
  _heroes = (NSMutableArray *) @"1 object"
    [0] = (APAWarrior *) 0x7b095b60
      APAHeroCharacter
  _goblinCaves = (NSMutableArray *) @"13 objects"
  _levelMap = (APADataMapRef) 0x12a46000
```

```
2014-08-04 10:34:24.115 Adventure[1509:21084] Cannot find
function pointer IOHIDLibFactory for factory
13AA9C44-6F1B-11D4-907C-0005028F18D5 in CFBundle/CFPlugIn
0x7b2708e0 </System/Library/Extensions/IOHIDFamily.kext/
Contents/PlugIns/IOHIDLib.plugin> (bundle, not loaded)
2014-08-04 10:34:24.167 Adventure[1509:21117] Loading world
tiles
2014-08-04 10:34:26.837 Adventure[1509:21117] Loaded all world
tiles in 2.669845 seconds
2014-08-04 10:34:26.984 Adventure[1509:21084] Building the world
2014-08-04 10:34:27.157 Adventure[1509:21084] converted 11965
collision blocks into 36 volumes in 0.008072 seconds
(lldb)
```

Swift Explorations

- The left window is the Variable Viewer which shows a drop-down list of all of your program's objects and instance variables held in memory at the current scope. We'll teach you how to review the information in this window.

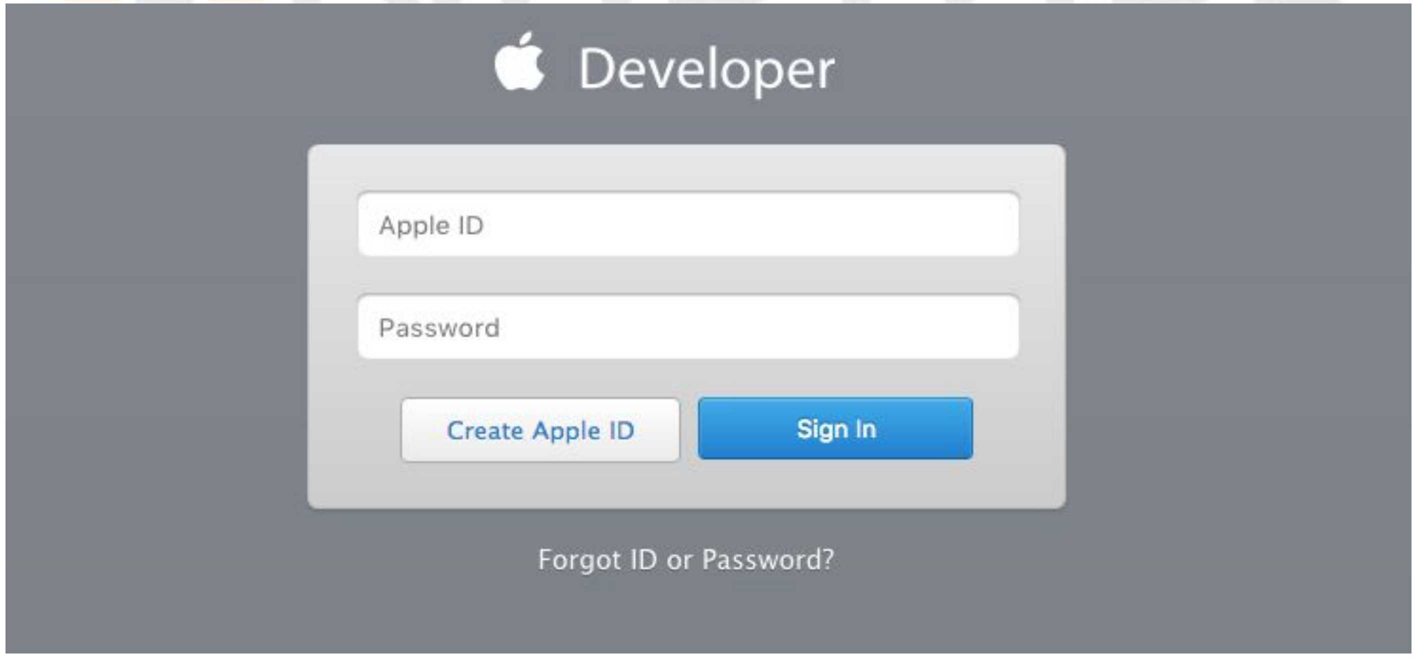
- The right window is the Console Output Viewer. This is where your NSLogs or Print will print out.

- We'll teach your more about how to use the debugger effectively in the next unit. For now, just understand the layout of the tools well enough to locate the Console Output viewer.

- If you can only see the Variable Viewer (or vice-versa), the Console Output Viewer is likely hidden. There's a pair of show/hide buttons in the bottom right corner of the Debug area which governs these two viewers.

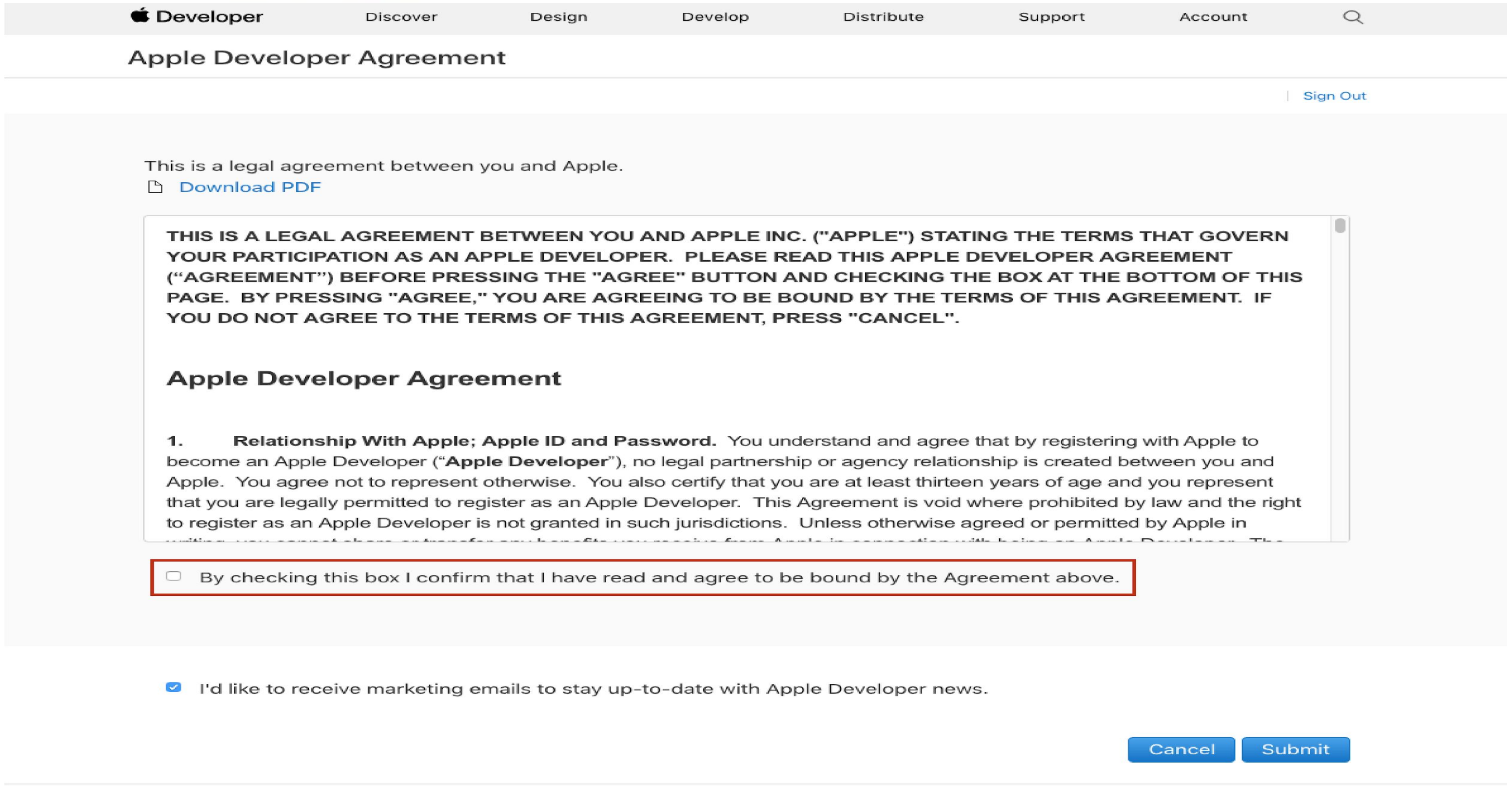
iOS Developer Program

1. First go to:https://developer.apple.com/enroll and click "Start your Enrolment'. Sign in using your Apple ID (or create an Apple ID if you do not have one yet).



Swift Explorations

2. Go over the Apple Developer Agreement, check the box confirming that you have read it and click on 'Submit'.



Apple Developer Agreement

This is a legal agreement between you and Apple.

THIS IS A LEGAL AGREEMENT BETWEEN YOU AND APPLE INC. ("APPLE") STATING THE TERMS THAT GOVERN YOUR PARTICIPATION AS AN APPLE DEVELOPER. PLEASE READ THIS APPLE DEVELOPER AGREEMENT ("AGREEMENT") BEFORE PRESSING THE "AGREE" BUTTON AND CHECKING THE BOX AT THE BOTTOM OF THIS PAGE. BY PRESSING "AGREE," YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, PRESS "CANCEL".

Apple Developer Agreement

1. **Relationship With Apple; Apple ID and Password.** You understand and agree that by registering with Apple to become an Apple Developer ("**Apple Developer**"), no legal partnership or agency relationship is created between you and Apple. You agree not to represent otherwise. You also certify that you are at least thirteen years of age and you represent that you are legally permitted to register as an Apple Developer. This Agreement is void where prohibited by law and the right to register as an Apple Developer is not granted in such jurisdictions. Unless otherwise agreed or permitted by Apple in

☐ By checking this box I confirm that I have read and agree to be bound by the Agreement above.

☑ I'd like to receive marketing emails to stay up-to-date with Apple Developer news.

Swift Explorations

3. Confirm that all your information is correct (email, name, location), then choose an entity. Please read Apple's explanation of entities below before you make a choice.

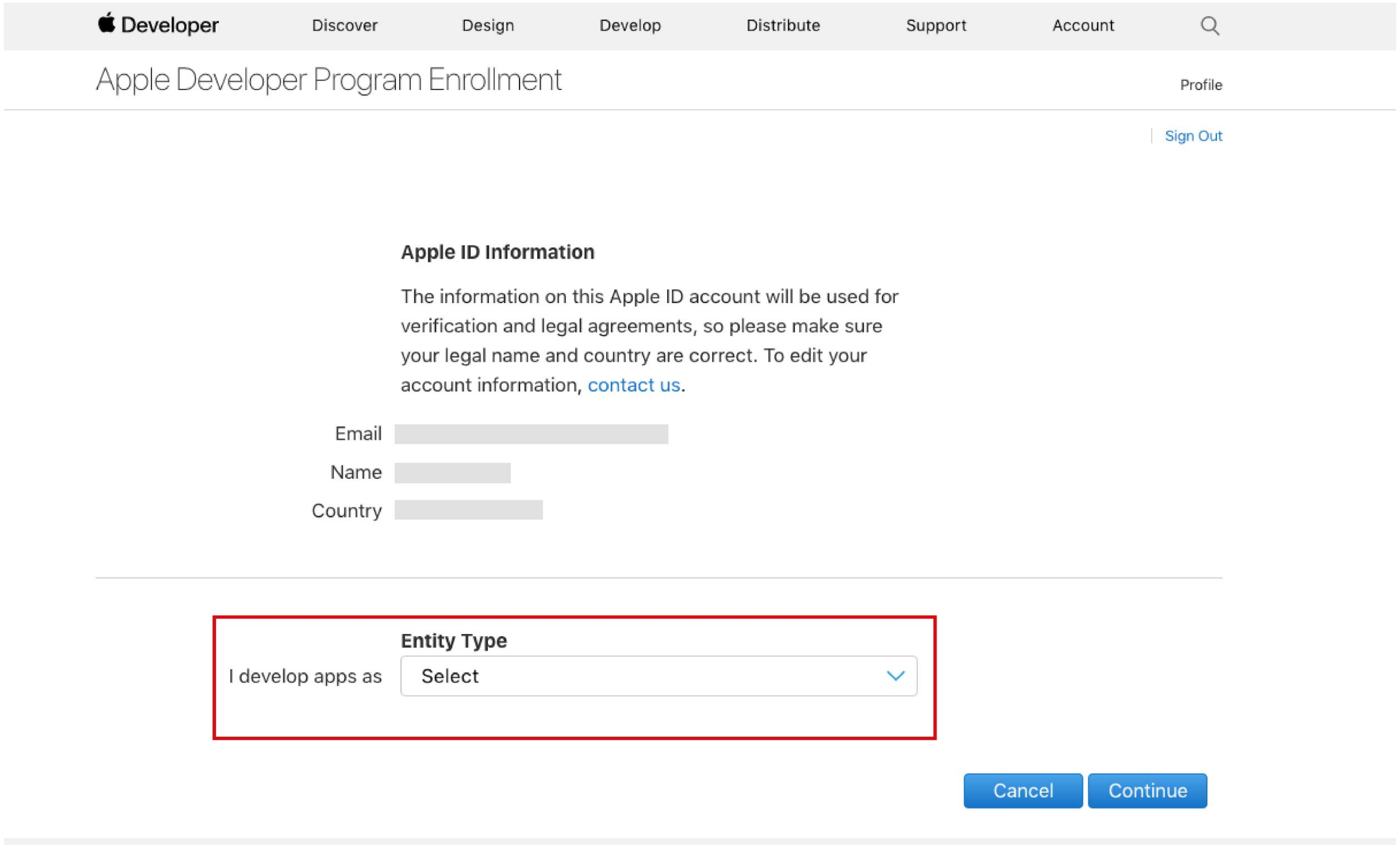
Explanation from Apple:

The Developer name listed on the App Store is based on the type of account selected from the options below. Apps published to Individual Developer Accounts will display the name listed on the iOS Developer Account. Apps published to Company/Organisation Developer Accounts will display the Company Name entered within the iOS Developer Account.

Entities:

- **Individual/Sole Proprietor/Single Person Business**: Individuals or companies without an officially recognised business (company without a Dun and Bradstreet number). This account only allows for a single primary login to be created to the iOS Developer Account
- **Company/Organisation:** Legally recognised Companies with a Dun and Bradstreet number can select this option. This will allow multiple user logins to be created and managed with varying permissions capabilities for each login. See here to check if your company is a legal recognised company with a Dun and Bradstreet (DUNS) number.

Swift Explorations



Apple Developer Program Enrollment

Apple ID Information

The information on this Apple ID account will be used for verification and legal agreements, so please make sure your legal name and country are correct. To edit your account information, contact us.

Email	
Name	
Country	

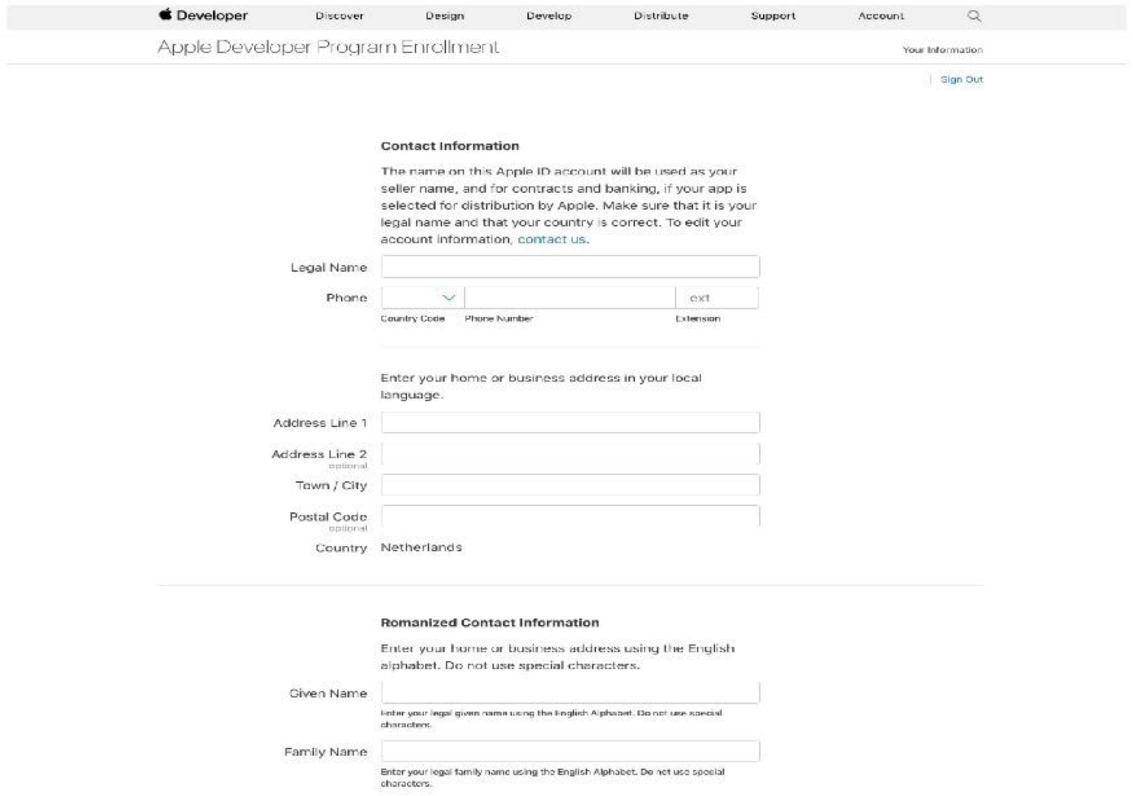
Entity Type

I develop apps as Select

Swift Explorations

4. Fill in all the contact details for your Developer Account.

5. Read the Apple Developer Program License Agreement, check the box confirming you have read the agreement and click ‘Continue'.



Apple Developer Program Enrollment

Your Information

Sign Out

Contact Information

The name on this Apple ID account will be used as your seller name, and for contracts and banking, if your app is selected for distribution by Apple. Make sure that it is your legal name and that your country is correct. To edit your account information, contact us.

Legal Name

Phone — Country Code — Phone Number — ext Extension

Enter your home or business address in your local language.

Address Line 1

Address Line 2 optional

Town / City

Postal Code optional

Country Netherlands

Romanized Contact Information

Enter your home or business address using the English alphabet. Do not use special characters.

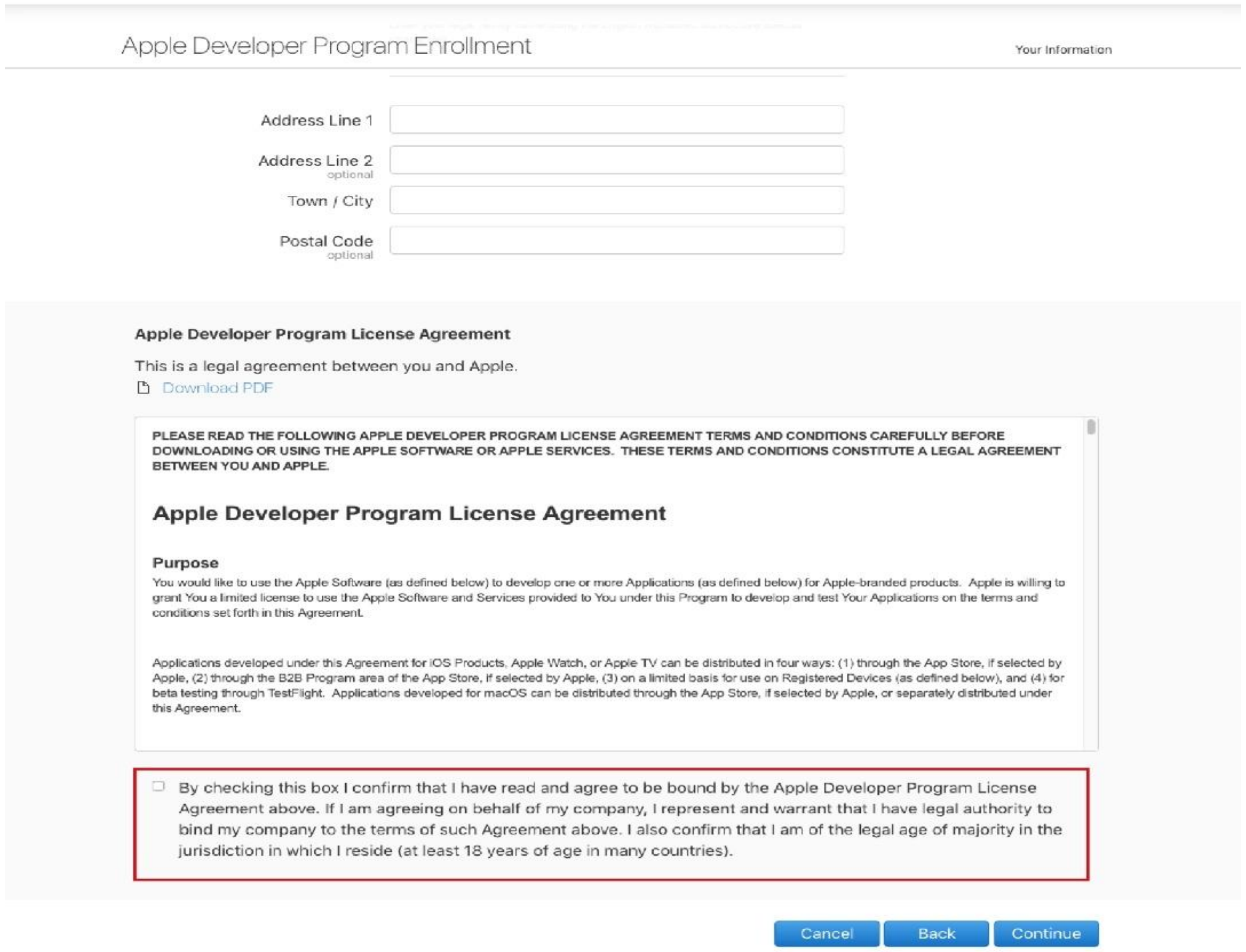
Given Name

Enter your legal given name using the English Alphabet. Do not use special characters.

Family Name

Enter your legal family name using the English Alphabet. Do not use special characters.

Swift Explorations



Apple Developer Program Enrollment

Your Information

Address Line 1	
Address Line 2 optional	
Town / City	
Postal Code optional	

Apple Developer Program License Agreement

This is a legal agreement between you and Apple.

Download PDF

PLEASE READ THE FOLLOWING APPLE DEVELOPER PROGRAM LICENSE AGREEMENT TERMS AND CONDITIONS CAREFULLY BEFORE DOWNLOADING OR USING THE APPLE SOFTWARE OR APPLE SERVICES. THESE TERMS AND CONDITIONS CONSTITUTE A LEGAL AGREEMENT BETWEEN YOU AND APPLE.

Apple Developer Program License Agreement

Purpose

You would like to use the Apple Software (as defined below) to develop one or more Applications (as defined below) for Apple-branded products. Apple is willing to grant You a limited license to use the Apple Software and Services provided to You under this Program to develop and test Your Applications on the terms and conditions set forth in this Agreement.

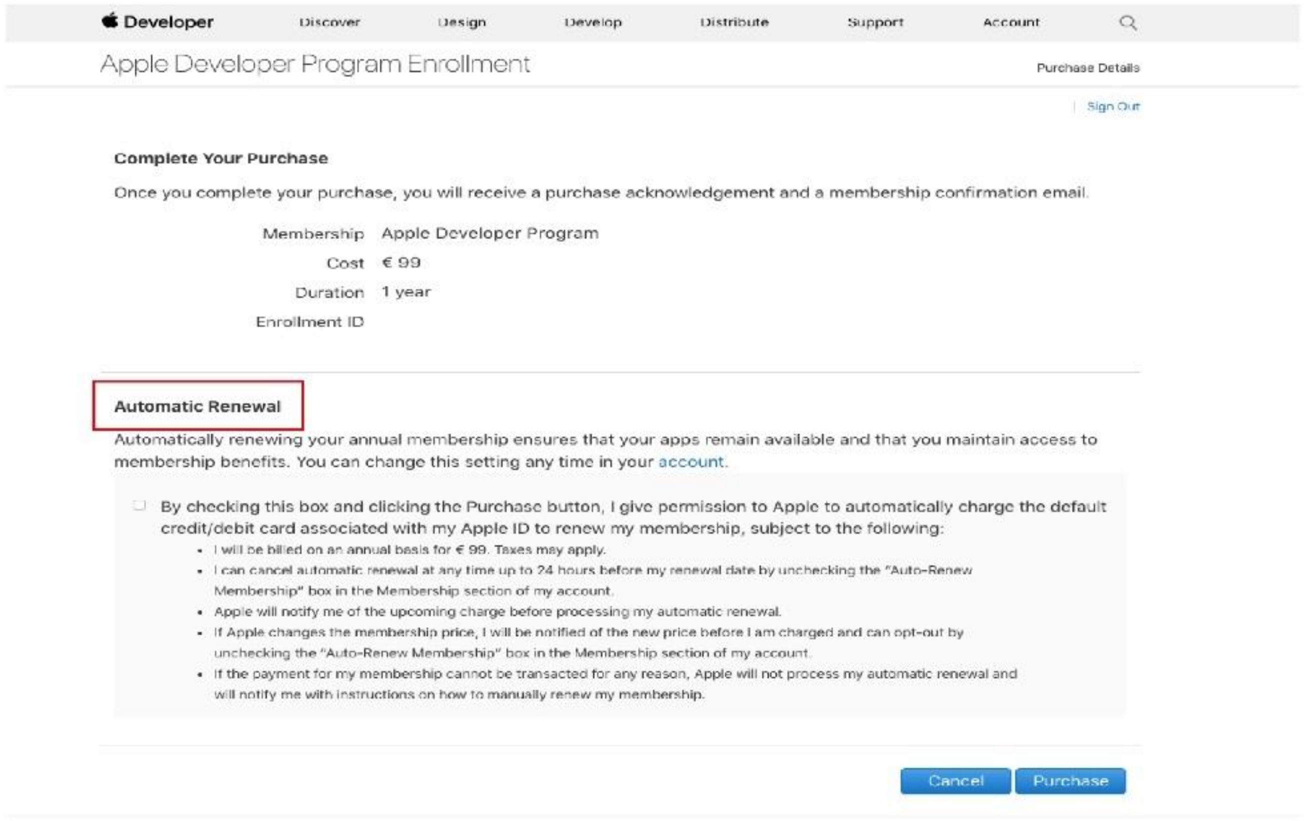
Applications developed under this Agreement for iOS Products, Apple Watch, or Apple TV can be distributed in four ways: (1) through the App Store, if selected by Apple, (2) through the B2B Program area of the App Store, if selected by Apple, (3) on a limited basis for use on Registered Devices (as defined below), and (4) for beta testing through TestFlight. Applications developed for macOS can be distributed through the App Store, if selected by Apple, or separately distributed under this Agreement.

☐ By checking this box I confirm that I have read and agree to be bound by the Apple Developer Program License Agreement above. If I am agreeing on behalf of my company, I represent and warrant that I have legal authority to bind my company to the terms of such Agreement above. I also confirm that I am of the legal age of majority in the jurisdiction in which I reside (at least 18 years of age in many countries).

Swift Explorations

6. Confirm that all the previously entered information is correct en click continue.

7. Click ‘Purchase’ to enrol and pay for your yearly iOS Developer Account. Don't forget to check the Automatic Renewal box if you want you membership to be automatically renewed.



Apple Developer Program Enrollment

Complete Your Purchase

Once you complete your purchase, you will receive a purchase acknowledgement and a membership confirmation email.

Membership	Apple Developer Program
Cost	€ 99
Duration	1 year
Enrollment ID	

Automatic Renewal

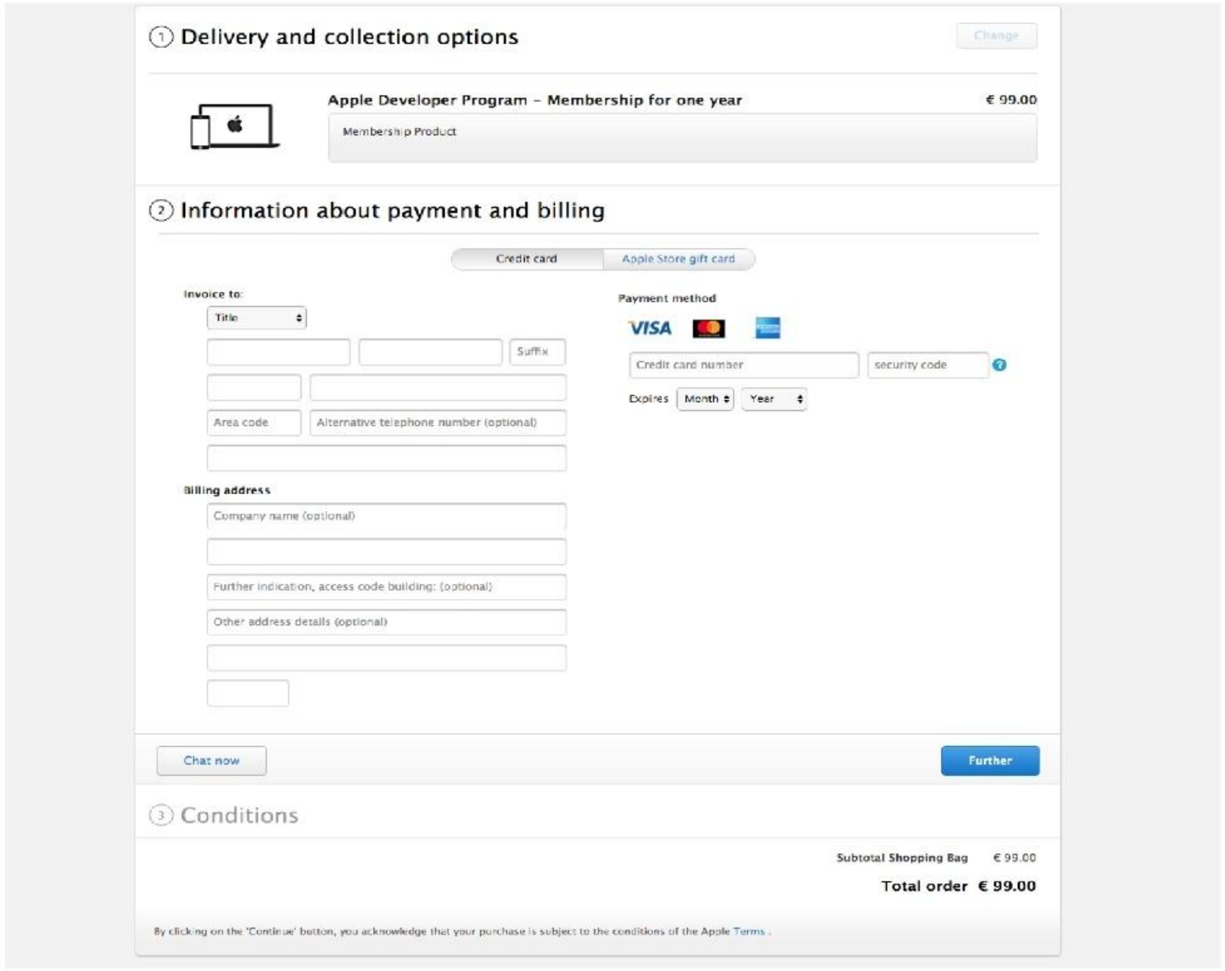
Automatically renewing your annual membership ensures that your apps remain available and that you maintain access to membership benefits. You can change this setting any time in your account.

☐ By checking this box and clicking the Purchase button, I give permission to Apple to automatically charge the default credit/debit card associated with my Apple ID to renew my membership, subject to the following:

- I will be billed on an annual basis for € 99. Taxes may apply.
- I can cancel automatic renewal at any time up to 24 hours before my renewal date by unchecking the "Auto-Renew Membership" box in the Membership section of my account.
- Apple will notify me of the upcoming charge before processing my automatic renewal.
- If Apple changes the membership price, I will be notified of the new price before I am charged and can opt-out by unchecking the "Auto-Renew Membership" box in the Membership section of my account.
- If the payment for my membership cannot be transacted for any reason, Apple will not process my automatic renewal and will notify me with instructions on how to manually renew my membership.

Swift Explorations

8. Sign in with your Apple ID and fill in your billing information. Then click continue to complete the setup of your iOS Developer Account.



1 Delivery and collection options

Apple Developer Program – Membership for one year € 99.00

Membership Product

2 Information about payment and billing

Invoice to:

Billing address

Company name (optional)
Further indication, access code building: (optional)
Other address details (optional)

Payment method

3 Conditions

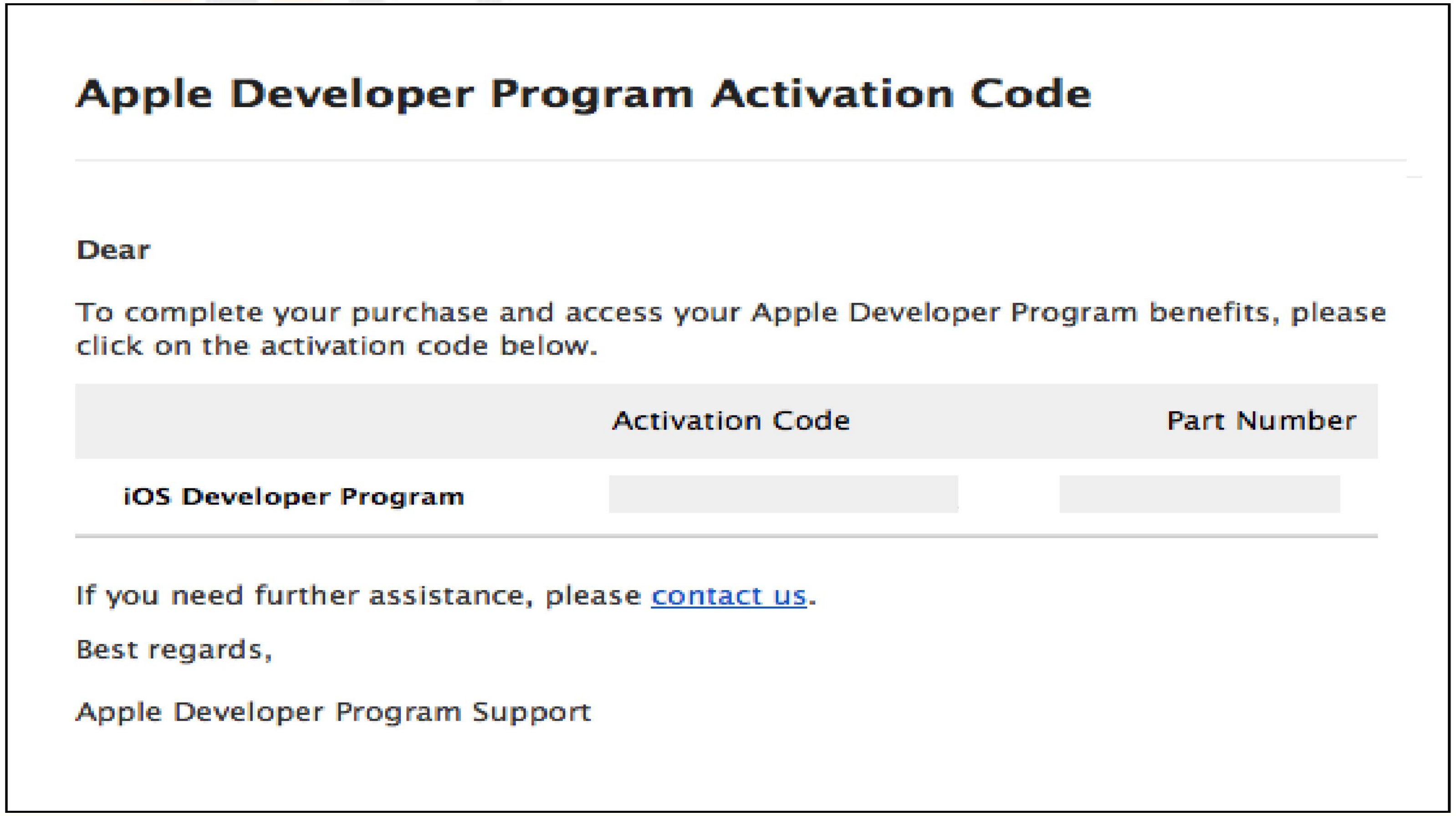
Subtotal Shopping Bag € 99.00

Total order € 99.00

By clicking on the 'Continue' button, you acknowledge that your purchase is subject to the conditions of the Apple Terms .

Swift Explorations

9. Fill in any remaining information on the following pages. Within 24 hours you will receive a confirmation email from Apple letting you know the setup of your iOS Developer Account was successful.



Apple Developer Program Activation Code

Dear

To complete your purchase and access your Apple Developer Program benefits, please click on the activation code below.

	Activation Code	Part Number
iOS Developer Program		

If you need further assistance, please contact us.

Best regards,

Apple Developer Program Support

Creating Apple iOS Developer Account

Step 1: Open any Browser, Go to https://developer.apple.com/

Step 2: Select Account Option

Step 3: Select Create Your's Now option

Step 4: Fill all the details i.e Name, Country/Region, Date of Birth, Mail id and Create Passwords, Security Questions, Enter Character's in Image and Select Continue option

Step 5: Apple will send you the activation mail and code

Step 6: After Entering the code then Apple iOS Developer Account will be created

Step 7: Now Sign-In to Developer Portal and then Access the Documents and Software's provided by Apple.

Creating and Testing Simple iOS Application in Simulator

Step 1: Open XCODE TOOL

Step 2: Select iOS Option

Step 3: Select Single-View Based Application Template, then Continue

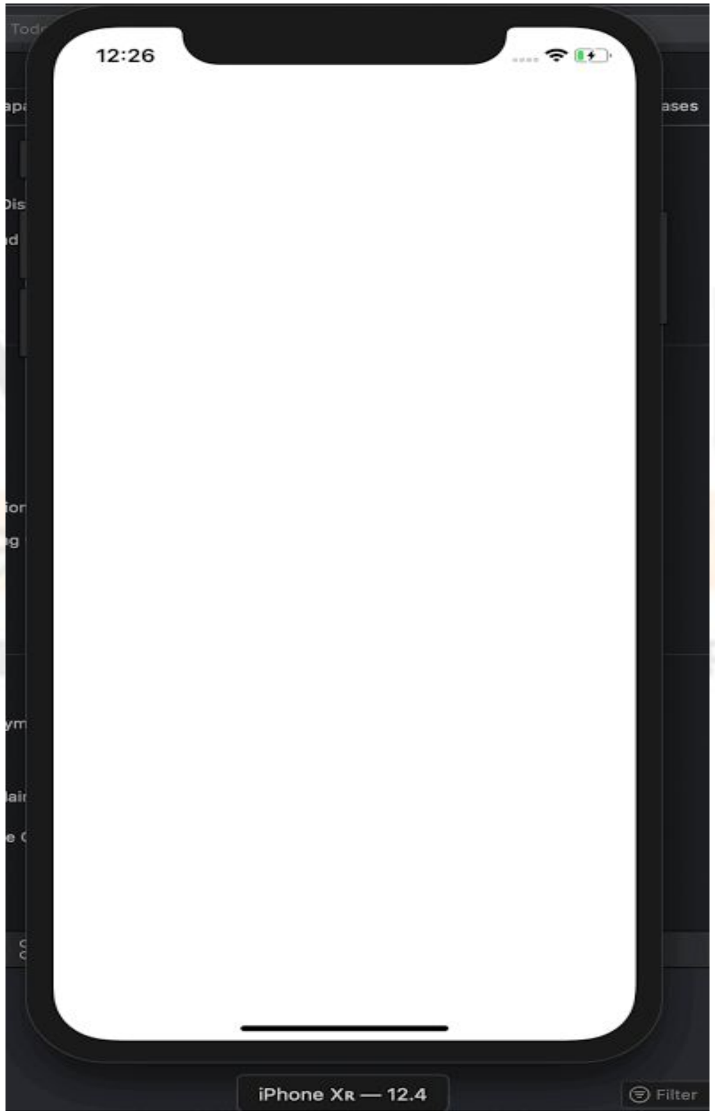
Step 4: Give the Product Name, Organisation Name, Organisation Identifier, Select Programming language as Swift, By default options will checked including Unit and UI TEST and Select continue

Step 5: Select the Path to save Project and Select Create Option

Step 6: Now Single-View iOS application Template will be created

Step 7: Select Play Option in iOS Application Window, then the out put will be Displayed in iOS Simulator & Select Stop Option in iOS Application window then Application will stop in running iOS Simulator

Swift Explorations



Assignment

1. Write a program to collect n names and print them using arrays.

2. Write a program to traverse array using fast enumerations.

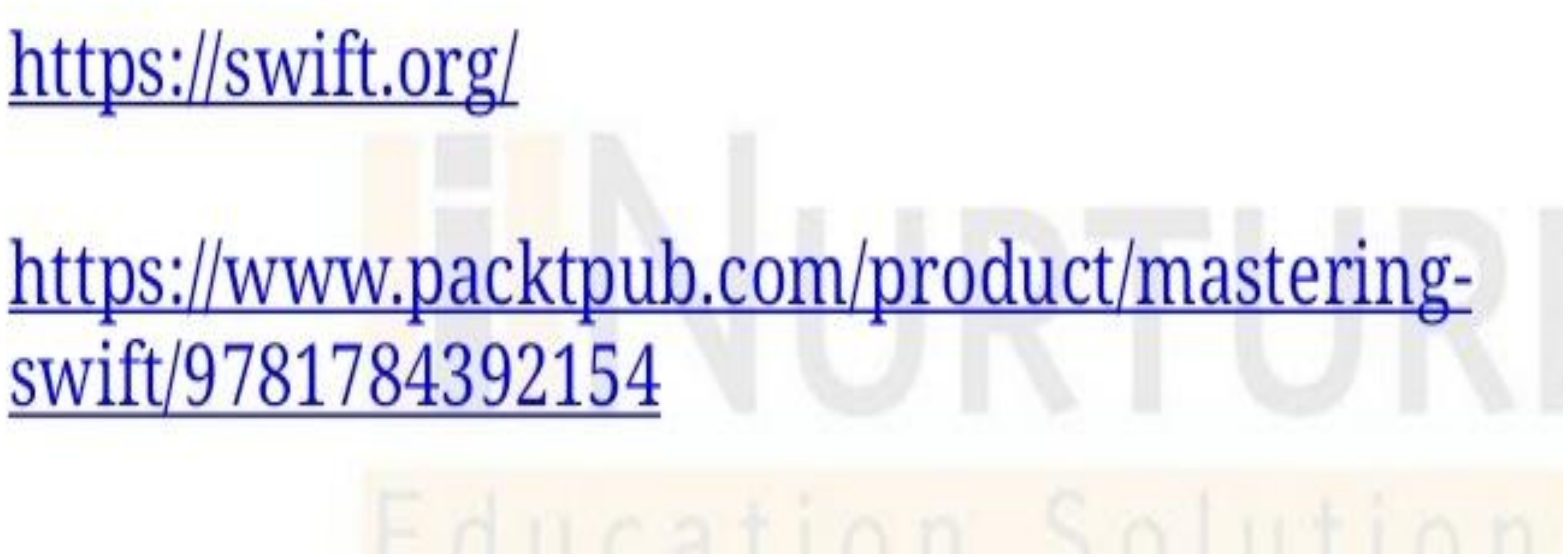
3. Write a program to call one function on another function using Calling Methods Syntax.

Swift Explorations

Document Links

Topics	URL	Notes
Swift Pragramming Language	https://swift.org/	Give the information to learn Swift Programming Language
Swift Notes pdf	Mastering Swift 5 Fifth Edition https://www.packtpub.com/product/mastering-swift/9781784392154	Mastering Swift 5 Topics

E-Book Links



https://swift.org/

https://www.packtpub.com/product/mastering-swift/9781784392154