

嵌入式系統設計

Embedded System Design

Lab 1: Swift Language Overview and iOS Basic GUI

蕭安紜助教製作

一. 實驗目的

了解Xcode使用方式，Swift的基本語法以及iOS的基本GUI使用方式

二. 實驗需求

環境：macOS 13.4 或以上

IDE：Xcode 14.0 或以上

語言：Swift 5

三. MacOS基本操作

複製：command⌘ + c

貼上：command⌘ + v

搜尋：command⌘ + f

上一步：command⌘ + z

儲存：command⌘ + s

切換中英文：caps lock

其餘指令可以參考：<https://support.apple.com/zh-tw/HT201236>

如果鍵盤不是蘋果自家的，Windows按鍵可以替代command⌘

- 指令將變成複製：Windows + c，以此類推

四. Swift 5 基本語法介紹

1. 名詞對照

- 變數：variable
- 函數：function
- 函數參數：parameter
- 函數引數：argument
- 型別：type
- 類別：class
- 類別的變數：property
- 類別的函數：method

2. 變數宣告

```
var i: Int = 0
```

- 完整的變數宣告，包含
變數名稱：i
變數型別：Int
變數初始值：0

```
var i = 0
```

- 省略變數型別，“i”之變數型別與“=”後面之“0”相同

3. 常數宣告

```
let pi: Double = 3.14159
```

- 完整的常數宣告，包含
常數名稱：pi
常數型別：Double
常數值：3.14159

```
let pi = 3.14159
```

- 省略變數型別，“pi”之變數型別與“=”後面之“3.14159”相同

4. 函數宣告

```
func appendStringWithNumber(string: String, number: Int) -> String
{
    let newString = string + String(number)
    return newString
}
```

- 函數名稱：appendStringWithNumber
函數參數數量：2
函數參數名稱：string, number
參數型別：String, Int
函數回傳型別：String

```
func printHello(name:String)
{
    print("hello, "+name+"!")
}
```

- 無回傳函數，等價於：

```
func printHello(name:String) -> Void
{
    print("hello, "+name+"!")
}
```

5. 函數呼叫

```
printHello(name: "Eric")
```

- 單一參數呼叫方法

```
appendStringWithNumber(string: "No.", number: 1)
```

- 多參數呼叫方法，注意，輸入方式為”函數參數名稱: 輸入變數”
- 在此，appendStringWithNumber 之第二個參數的名稱為 number，所以()內第二塊要填“number: 1”

6. 引數標籤與函數名稱

使用引數標籤讓在call函數時看到對parameter的敘述

```
func append(aString string: String, withANumber number: Int) -> String
{
    let newString = string + String(number)
    return newString
}

append(aString: "123", withANumber: 456)
```

- 函數名稱：append
函數參數數量：2
函數參數名稱：string, number
引數標籤：aString, withANumber
參數型別：String, Int
函數回傳型別：String

若parameter的敘述已寫在function name裡面，可使用 _ 來省略引述標籤

```

func appendString(_ string: String, withANumber number: Int) -> String
{
    let newString = string + String(number)
    return newString
}

appendString("123", withANumber: 456)

```

- 函數名稱：appendString
函數參數數量：2
函數參數名稱：string, number
引數標籤：withANumber
參數型別：String, Int
函數回傳型別：String

7. Class 與 繼承

```

class ViewController: UIViewController
{

    var name: String = "Eric"

    override func viewDidLoad() {
        super.viewDidLoad()
        self.printName()
    }

    func printName()
    {
        print(self.name)
    }
}

```

- 宣告一個名叫 ViewController 的 Class，其繼承 UIViewController 這個 Class
- ViewController 包含一個名為 name 的 property，型態為 String
- override 表示此 method 取代 parent class 原有的 method
- super 表示 parent class，通常在取代 parent class 函數時，保險起見會先 call parent class 的同一個函數，以免遺漏 parent class 本來該做的動作
- 在 class 內呼叫自身的 method 或是 property 時，要加 self。self.name 及 self.printName() 表示指定此

8. String

在Swift中，可用不同的方式將多個變數組成一個新的 String

- String Concatenating：利用 + 號連接 String

```
let newString = "i = "+String(i)+" , pi = "+String(pi)
```

- String Interpolation(建議)：利用\()嵌入其他型別物件

```
let newString = "i = \(i), pi = \(pi)"
```

- Formatted String : 與c語言類似的formatted string用法

```
let newString = String(format: "i = %d, pi = %f", i, pi)
```

9. 命名規則

- 專案名稱大寫開頭
- 變數/函數名稱小寫開頭，單字首大寫
 - myString
 - myString.containsString("s")
- 變數類別(Class Name) 開頭大寫
- 變數名稱盡量完整表達參數用途、型別
- 函數名稱+引數標籤+參數名稱為描述函數用途的句子
 - myString.components(separatedBy:"s")

10. Interface Builder attributes

@IBOutlet :

Interface Builder內的物件可連結至這個變數

```
@IBOutlet var text : UITextField!
```

- Interface Builder 內可有 UITextField 連結到 text 這個變數

@IBAction :

Interface Builder 中的動作可連結至這個參數

```
@IBAction func sendClicked(sender : UIButton)
```

- Interface Builder 中 UIButton 類別的物件可將其動作連結至 sendClick 這個函數
- 發動這個動作的 UIButton 本身會被當作參數傳入 sendClicked 函數當作 sender

五. Optional

Swift 特有的型態 Optional，表示此數值可以有值，也可以是nil，如果變數是 optional的型態，例如 optional String，則必須要先把此變數 unwrap 回 String 才能用 String 的 功能。

1. 宣告

```
var optString1: String?  
var optString2: String? = nil  
var optString3: String? = "123"  
var optDouble: Double?
```

`String?` 就是一種型態的名稱，請把?跟String視為一體。

以上的`optString1`, `optString2`, `optString3` 的資料型態都是 `String?`

(optional string)，而`optDouble`的型態為 `Double?` (optional double)。

如果是optional，宣告時可以不用給初始值，會直接設定成nil。

2. Unwrap

要把真的有值的 optional 變數取出來用，或是判斷 optional 變數是否為nil，有以下方法。

(1)Forced Unwrapping

```
var string: String = optString3!
```

此為不安全的用法，加上"!"後，會將optional string強制轉回string，但若原本的optional string沒有值，也就是nil，runtime會跳錯誤(exception)。

(2)Optional Binding

```
if let unwrappedString = optString1 {  
    print("has value \(unwrappedString)")  
} else  
{  
    print("no value")  
}
```

如果optString有值，則將wrappedString設定成那個值，進入true的block，若沒有則進else，wrappedString只有在true的block有效。

let可以連用，檢查多個optional變數：

```
if let unwrappedString1 = optString1, let unwrappedString2 = optString2
{
    print("has value \(unwrappedString1) and \(unwrappedString2)")
}
```

(3)Guard let

Guard 與 if 用法很像， guard為 Swift 的 Early Exit，必須與Function一起用，如果Guard沒過，則不會執行Guard之後的function的code。

```
func addOne(number:Int?) -> String
{
    guard let unwrappedNumber = number else { return "no value" }
    var newValue = unwrappedNumber + 1
    return "\(newValue)"
}

print(addOne(number: nil))
print(addOne(number: 3))
```

Guard 與 if 不同之處在於，unwrap過後的變數，可以在後續繼續使用。

* 不能直接用if(optString1)判斷是不是nil

* if let, guard let 也可以換成用 if var 或 guard var

3. 特殊用法

(1)Implicitly Unwrapped Optionals

如果在宣告變數時，用了Optional，會導致日後要使用這個變數時要一直做unwrapping，如果很確定此變數在宣告過後，很快會被設定值，且會一直有值，那就可以使用Implicitly Unwrapped Optionals來宣告變數。

```
let assumedString: String! = "An implicitly unwrapped optional string."
let implicitString: String = assumedString // no need for an
exclamation mark
```

`String!` (implicitly unwrapped optional)也是一種型態名稱，跟 `String?` 一樣，我們可以把implicitly unwrapped optional視為會自動unwrap的 optional變數。

如果要取用implicitly unwrapped optional的值，但他是nil時，會跳 runtime error。

(2)Optional Chaining

如前面所述，想取用optional物件裡的值的property或method時，若使用forced unwrapping，有可能會因為物件是nil而導致調用不到nil的 property或method而導致runtime error。此時，可使用optional chaining，用法與forced unwrapping很像，差異在於，如果物件是 nil，會阻擋之後的property或method的調用，直接回傳nil。

```
var button: UIButton? = nil  
  
var a = button!.titleLabel!.text  
var b = button?.titleLabel?.text
```

上面的例子中，optional 的 button (`UIButton?`) 擁有optional 的 `titleLabel` (`UILabel?`)。若想要取得button的文字，用a的方式跟b的方式都可以compile。但只要 button 或 `titleLabel` 有任一是nil，則a會出現 runtime error；用b的方式，則b會得到nil。

(3)Nil-Coalescing Operator

```
var string = optString1 ?? "no value"
```

等價於

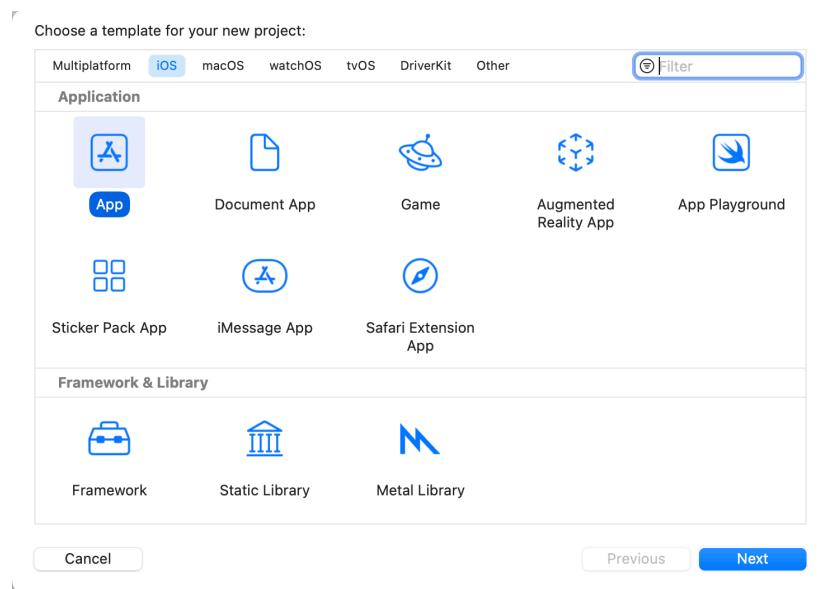
```
var string = (optString1 != nil) ? optString1! : "no value"
```

Nil-Coalescing (`??`) 可用於在取用optional物件時，給nil的狀況一個預設的值。即為若`optString1`為nil時，`string`會得到"no value"的值而不會得到nil。

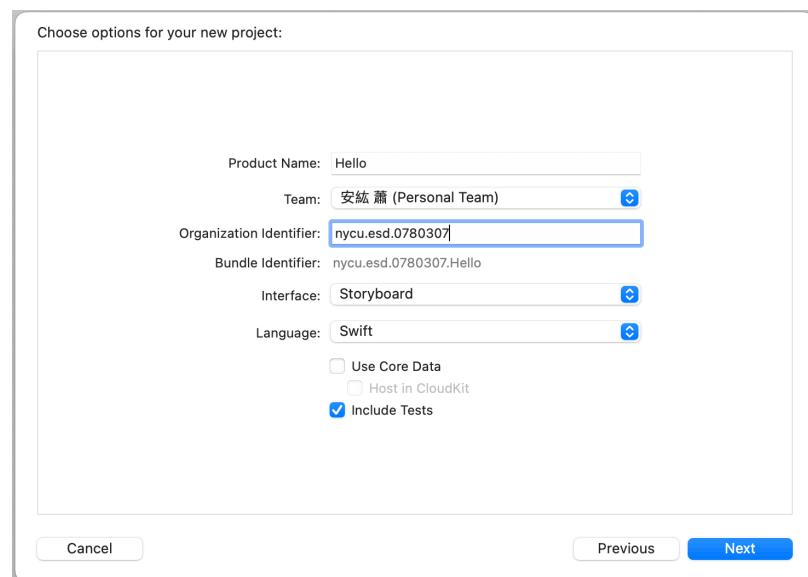
六. 實驗步驟

4. 建立 iOS 專案

- (1) 開啟Xcode，左上角menubar，選擇 File -> New -> Project
- (2) 左方選擇 iOS 內的 Application ，右方選擇 Single View Application ，按 Next



- (3) Product Name 輸入 Hello Organization identifier 輸入 nycu.esd."你的學號"
Language 選擇 Swift 按下 Next，存在桌面



- (4) 點選左方 Project navigator 中的 ViewController.swift，可看到以下程式碼

The screenshot shows the Xcode interface with the following details:

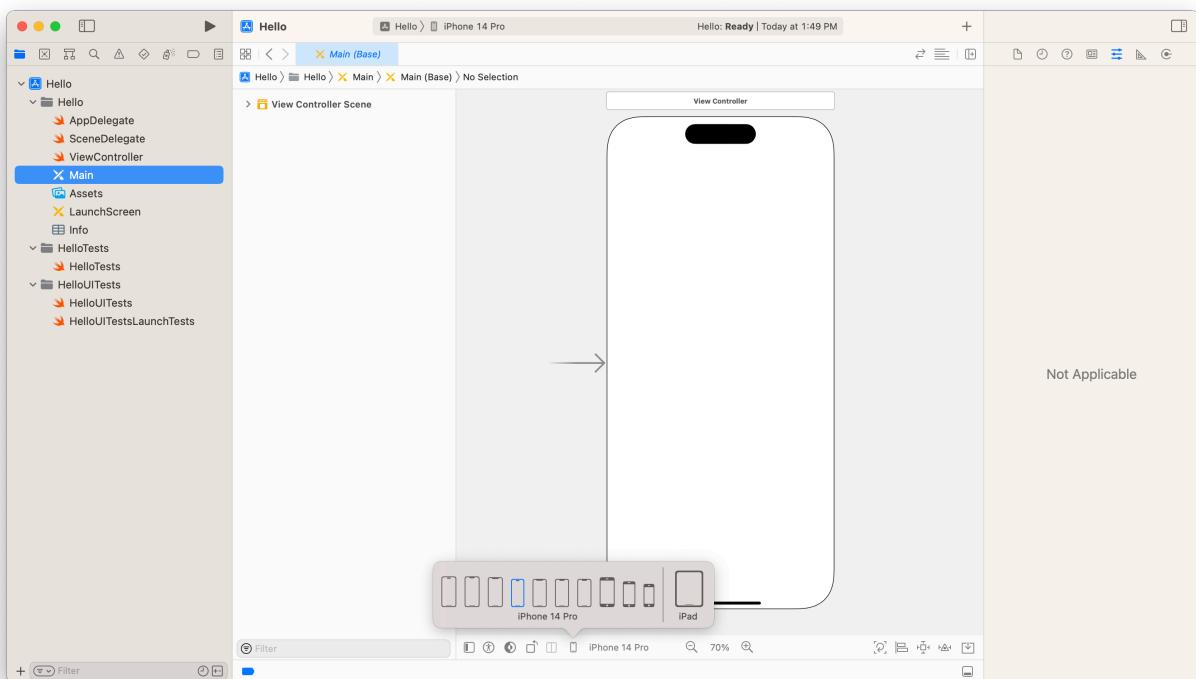
- Title Bar:** Hello Ready | Today at 1:58 PM
- Project Navigator:** Shows the project structure with files like AppDelegate.swift, SceneDelegate.swift, ViewController.swift, Main.storyboard, Assets.xcassets, LaunchScreen.storyboard, Info.plist, and various test files.
- Editor:** Displays the code for ViewController.swift:

```
1 //  
2 // ViewController.swift  
3 // Hello  
4 //  
5 // Created by An on 2024/1/12.  
6 //  
7  
8 import UIKit  
9  
10 class ViewController: UIViewController {  
11     override func viewDidLoad() {  
12         super.viewDidLoad()  
13         // Do any additional setup after loading the view.  
14     }  
15 }  
16  
17  
18 }  
19  
20 }
```
- Utilities Area:** Includes sections for Identity and Type (Name: ViewController.swift, Type: Default - Swift Source), Location (Relative to Group), View Controller.swift, Full Path (/Users/an/Documents/ESD/2024 Lab>Hello>Hello/ViewController.swift), On Demand Resource Tags, Target Membership (Hello checked, HelloTests and HelloUITests unchecked), Text Settings (Text Encoding: No Explicit Encoding, Line Endings: No Explicit Line Endings), and Indent Using (Spaces checked, Widths: Tab 4, Indent 4, Wrap lines checked).

一個 View Controller 可以控制App的一個頁面。我們先在 Storyboard 中先拉好這頁使用者會用到的 GUI 物件，接下來在 View Controller 中用程式碼控制這些物件的作用、互動。

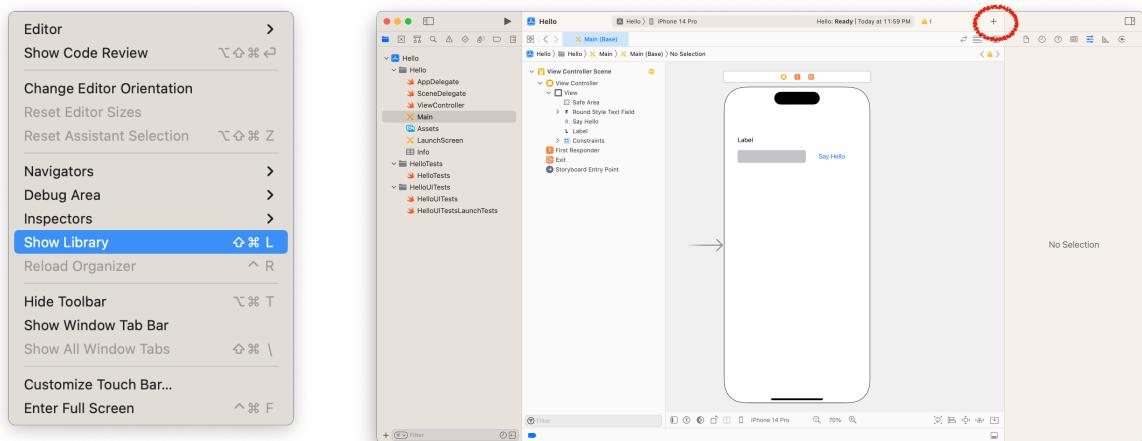
5. Xcode Interface Builder 與程式 GUI Layout

(1) 點選專案左方 Project navigator 中的 Main.storyboard 開始編輯程式使用者界面

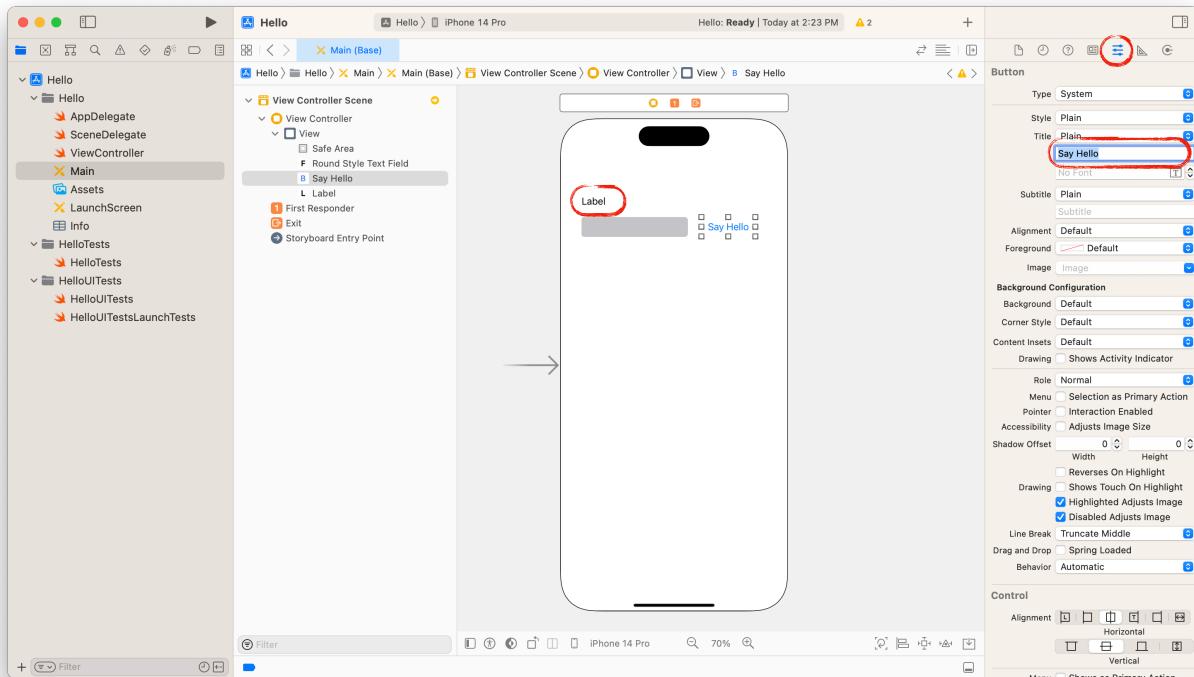


(2) 如圖，選取ViewController對應到的介面(黃色圈圈)，並在下方將其介面大小改成適當大小，方便layout

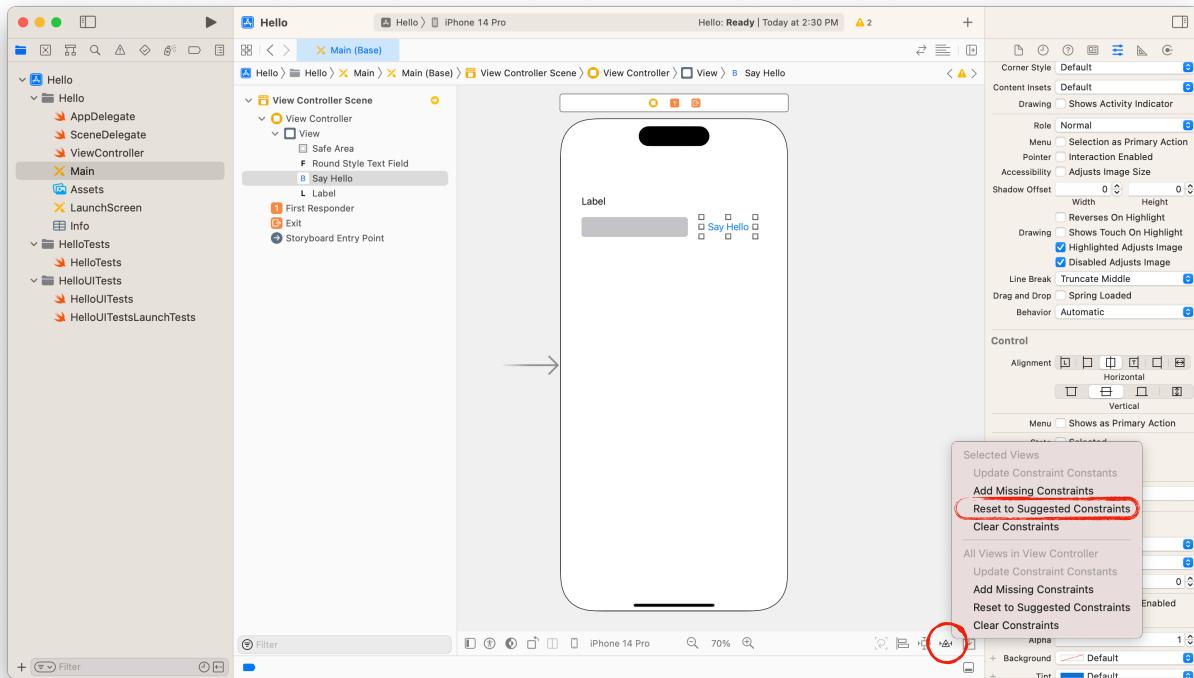
(3) 從menu bar->view->Show Library 中找到 Text Field (UITextField) 、 Button (UIButton) 、 Label (UILabel) 三種物件，拉入畫面中，排成想要的排列。



(4) 點選Button物件，利用右方 Attributes inspector 將按鈕名稱改成 Say hello ，同樣方式，將 Label 的預設內容刪除。

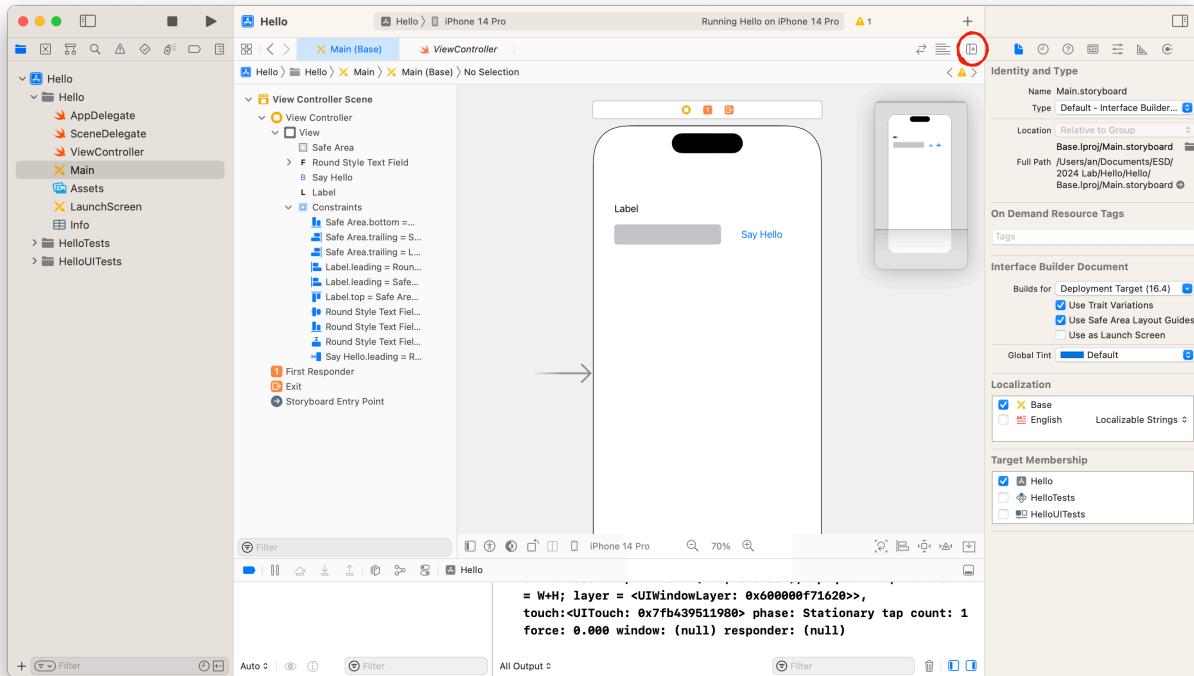


(5) 固定物件 Layout 。點選右下方 Resolve Auto Layout Issue -> All Views in View Controller -> Reset to Suggested Constraints

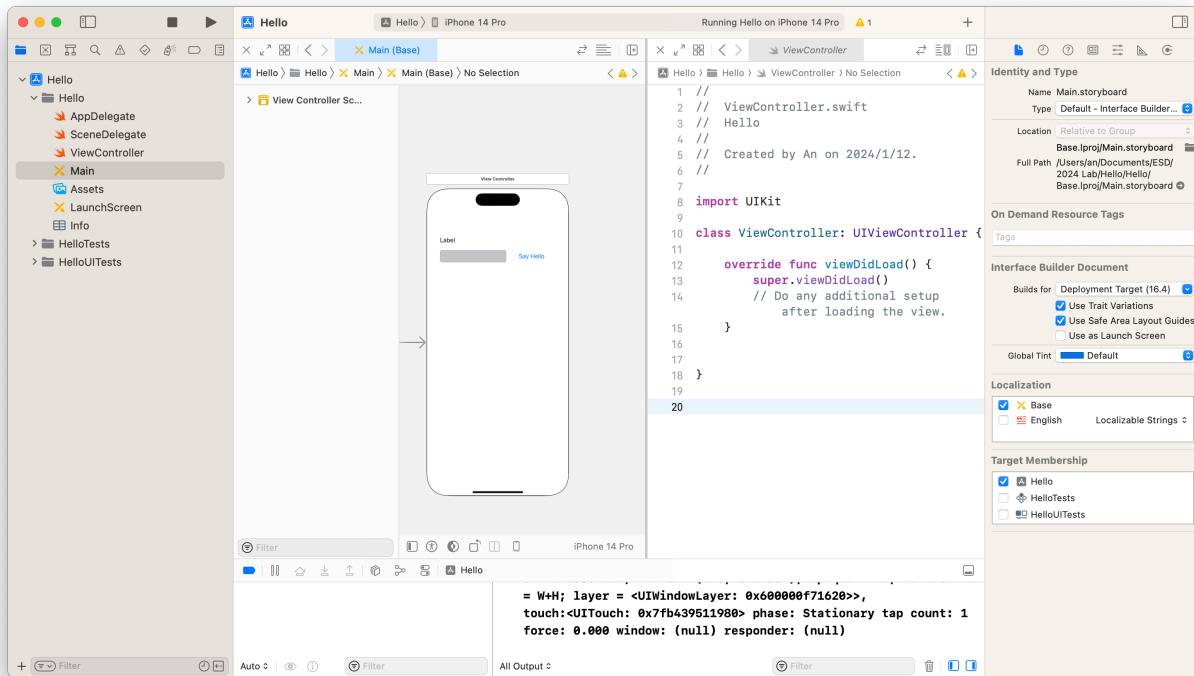


6. GUI 與 Code 的連結

(1) 點選右上Add Editor on Right來新增視窗，並使新視窗顯示ViewController

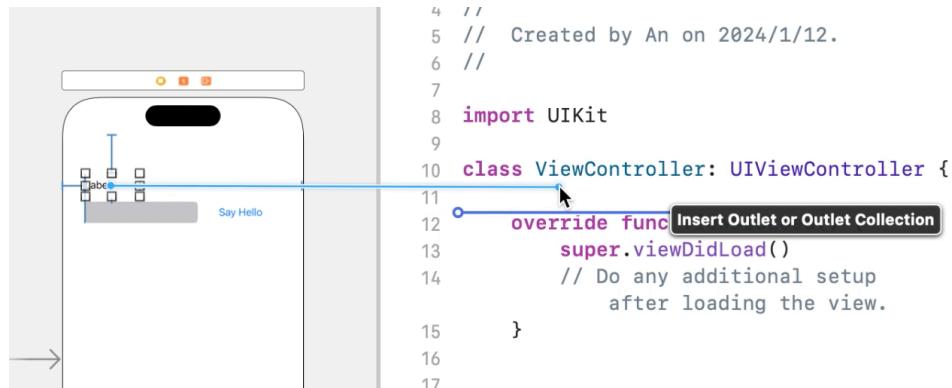


使整個介面呈現如下圖所示



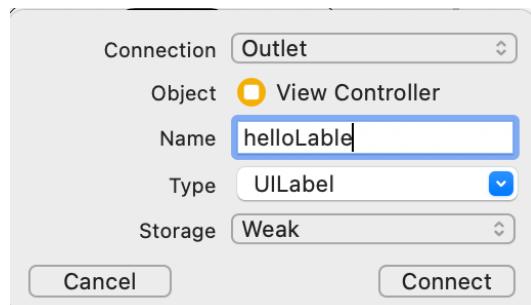
(2) 將 GUI 物件連結到 ViewController 的 Property (IBOutlet)。在左方 Interface Builder 中選取 Label ，按住control的同時用滑鼠右鍵拖移致右方的程式碼中

“class ViewController: UIViewController {“ 與 “override func viewDidLoad() {“ 的中間。



(3) 在彈出的框框中如以下設定，按下 Connect

- Connection: **Outlet**
- Name: **helloLabel**
- Type: **UILabel** (預設值，確認是否為物件的Class)
- Storage: **Weak** (預設值)



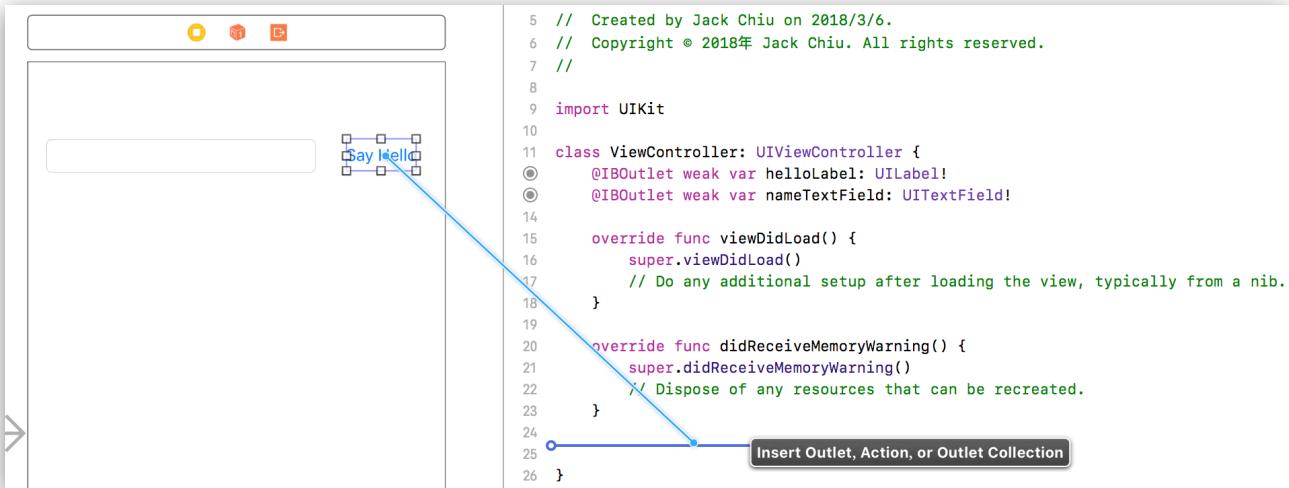
(4) 我們在介面上拉出的 Label 即會連結到 ViewController 中名為 helloLabel 的 Property 上。在 ViewController 中，self.helloLabel 就代表左方介面中剛剛連結的那個 UILabel 物件

```
1 //  
2 // ViewController.swift  
3 // Hello  
4 //  
5 // Created by An on 2024/1/12.  
6 //  
7  
8 import UIKit  
9  
10 class ViewController: UIViewController {  
11  
12 @IBOutlet weak var helloLabel: UILabel!  
13 override func viewDidLoad() {  
14     super.viewDidLoad()  
15     // Do any additional setup after loading the view.  
16 }  
17  
18  
19 }  
20  
21
```

(5) 將 Interface Builder 中的 Text Field 利用同樣方式連結到 ViewController 的 property，名為 nameTextField

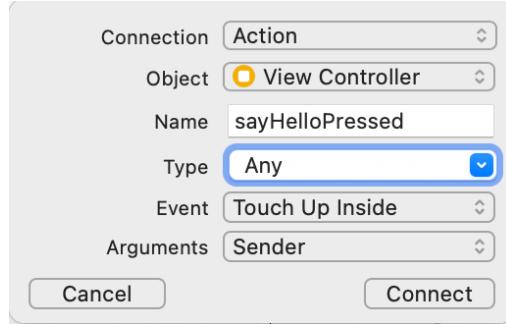
```
9 import UIKit
10
11 class ViewController: UIViewController {
12     @IBOutlet weak var helloLabel: UILabel!
13     @IBOutlet weak var nameTextField: UITextField!
14
15     override func viewDidLoad() {
16         super.viewDidLoad()
17         // Do any additional setup after loading the view, typically from a nib.
18     }
19
20     override func didReceiveMemoryWarning() {
21         super.didReceiveMemoryWarning()
22         // Dispose of any resources that can be recreated.
23     }
24
25
26 }
```

(6) 將 GUI 物件的事件連結到 ViewController 的 method (IBAction)。在左方 Interface Builder 中選取 “Say hello” Button，用滑鼠右鍵拖移致右方的程式碼中最後的 ”}” 上方。



(7) 在彈出的框框中如以下設定，按下 Connect

- Connection: Action
- Name: sayHelloPressed
- Type: AnyObject (預設值)
- Event: Touch Up Inside (預設值，可改成你需要的Button動作)
- Arguments: Sender (預設值)



- (8) 此時我們在介面的“Say hello”button若有 Touch Up Inside (點擊) 的事件發生時，會自動呼叫 ViewController 中名為 sayHelloPressed 的 method。“Say hello”button 會將自己本身當作 method 的參數傳入，所以在 sayHelloPressed function 中 sender 即代表程式介面上觸發事件(被點擊)的那個“Say hello”button。

7. 在 View Controller 設計程式流程與處理介面的互動

- (1) 返回 ViewController 進行編輯
- (2) 在 sayHelloPressed 中加入以下程式碼：

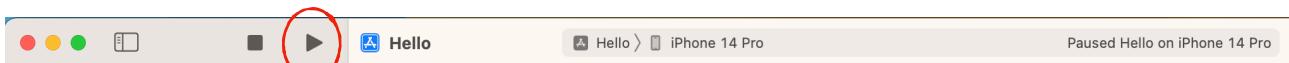
```
//將在nameTextField的內容合成新的字串  
let helloString = "Hello \(nameTextField.text!)!"  
  
//設定helloString字串到  
helloLabel.text = helloString  
  
//清空nameTextField輸入框  
nameTextField.text = ""
```

8. 建置、執行與測試專案

- (1) 在左上方點選 Set the active scheme ，選擇想要建置且執行的目標，在此選擇任一 iOS Simulator 。



- (2) 點選左上方 Build and then run the current scheme ，來建置且執行專案



- (3) 點選左上方 Stop the running scheme or application ，來停止執行中的程式



9. 執行結果

- (1) 在 Text Field 輸入文字後，按下 Say hello 按鈕，會將輸入內容合成歡迎訊息，顯示在Label上

七. Demo

請 Demo 實驗步驟最後的執行結果。

