National Yang Ming Chiao Tung University, Department of Electrical Engineering

嵌入式系統設計
Embedded System Design

Lab 2: Swift Control Flow and View Controller

蕭安紘助教製作

一. 實驗目的

了解 Swift 中常用的 Control Flow 以及 基礎的 View Controller 概念，試著時做出簡單的計算機App.

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. Swift Special Operators

1. 複習 Optional Syntax : (? , !)

宣告中的 ? , ! ：

```
var optString1: String?
```

- optString1 為一 **Optional** 的 string ，可以存 string 值，也可以存nil ，需要 unwrap 才能當 string 用。

```
var optString2: String!
```

- optString2 為一 I**mplicitly Unwrapped Optional** string，(可當成事先被 unwrap的 optional)，初始時可以是 nil，但馬上會被填入值，且之後就會保持有值得狀態，不會變回 nil 。 不用 unwrap ，可直接當 string 用，但如果取用時值是 nil 會有 runtime error。

範例：

```
var normalString:String
normalString = optString1!  //force unwrap
normalString = optString2
```

語句 (Statements) 中的 ? , ! ：

```
normalString = optString1!
```

- **Forced Unwrapping** ，將 optional 強制轉型回原本的型態，如果值為 nil 會 runtime error。

```
var button: UIButton?
var b = button?.titleLabel?.text
```

- **Optional Chaining**，碰到 ? 會先檢查 ? 前一個query return 的值是不是 nil，如果是 nil，則停止後續的query，避免使用 nil query的 runtime error。

2. Nil-Coalescing Operator (??)

```
a ?? b
```

- 等價於：

```
a != nil ? a! : b
```

- 省略變數型別， ”i ” 之變數型別與 ”=“ 後面之 ”0“ 相同。

3. Range Operators (a...b)

```
for index in 1...5 {
    print("\(index)")
}
/*
 1
 2
 3
 4
 5
*/
```

- For 迴圈會跑5次， index 分別為 1, 2, 3, 4, 5。

4. Half-Open Range Operator (a..<b)

```
let array = ["a", "b", "c", "d"]
let length = array.count
for i in 0..<length {
    print("i = \(i), element = \(array[i])")
}
/*
 i = 0, element = a
 i = 1, element = b
 i = 2, element = c
 i = 3, element = d
*/
```

- i 會從 0, 1, 2, 3 到 length -1，iterate 表單很好用。

5. One-Sided Ranges ([...a] , [a...])

```
print("\(array[...2])")
/*
["a", "b", "c"]
*/

print("\(array[2...])")
/*
["c", "d"]
*/
```

- 從一開始到 index a 的 element，從index a開始到最後的 element。
- 可以的用法如下：

```
for notFirst in array[1...] {
    print(notFirst)
}
/*
 b
 c
 d
*/
```

- 從 index 1 開始印出所有之後的 array elements。

四. Swift Control Flow

1. For-In Loops

```
let array = ["a", "b", "c", "d"]

for elemet in array
{
    print(elemet)
}
/*
 A
 b
 c
 d
*/
```

- element 會疊代過每個 array 裡的物件，有沒有順序性是依照 in 後面的物件型態，如果是 set 或是 dictionary 就不會保證是固定順序。

2. repeat-while

```
var i = 0

repeat{
    i += 1
    print("\(i)")
}while (i<4)

/*
 1
 2
 3
 4
*/
```

- repeat-while 與 do-while是一樣的。

3. Switch

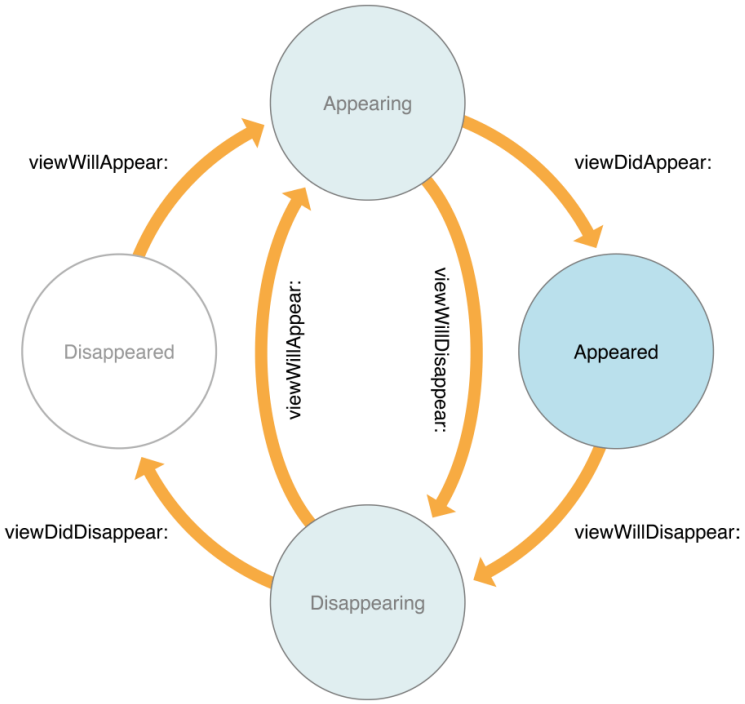
```
let operatorString = "+"
var result = 0

switch operatorString {
case "+":
    result = 1+1
case "-":
    result = 1-1
case "*", "/":
    result = 1
default:
    print("Other operator")
}
```

- 不需要break，只會進到 "第一個" 符合的case。
- case: 裡一定要有 statement ，不能是空的，default: 也不能是空的。
- 多個 case 用 "," 隔開。
- 可以使用 break：

```
default:
    break
```

五. View Controller and User Interface

1. View Controller Life Cycle
   - 通常，Storyboard 裡的一個 app 螢幕介面，會由一個 view controller 控制。
   - View controller 負責如何把資料呈現在介面(storyboard)的view上，也可以控制其內容所有 view 物件的 life cycle，例如：生成新的 view 加到介面上或從介面上移除 view 物件。
   - 同時，也可以定義這個介面的互動行為，例如：按下按鈕後發生的行為、切換到不同介面、螢幕被旋轉後相應要做什麼動作。
   - 在 iOS 載入一個新的介面到螢幕上時，在不同的 life cycle 時間點，會自動 call view controller 相對應的 method。
   - Override 這些 method，就可以讓 view controller 定義在某個 life cycle 時間點要做的對應動作。



   - 常用的 view controller life cycle call back:
     (1) viewDidLoad()：當此介面一從 storyboard 被創立好後會call這個 method ，此時，code 跟 UI 連接的 outlet (IBOutlet) 物件已經可以取得。
     (2) viewWillAppear()：介面要出現在螢幕前會先 call 此 method，可以在此做一些螢幕尚未出現前要做的事，例如：需要在介面出現後淡入某區塊的文字，可在此 method 內先把文字的透明度設成0，讓它剛出現在螢幕上是看不到的狀態。

(3) viewDidAppear()：當介面一出現在螢幕上時會 call 此 method，可以在此 method 內做介面出現後要做的動作，例如：前述的淡入文字，可在此用動畫將透明度0的文字區塊調成透明度1，以達到淡入的效果。

2. Interface Builder attributes (複習)

@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。
- 型態為implicitly unwrapped optional，因為此物件實際上是在storyboard生成的，並不是在 code initial的。在 view controller 剛產生的時候，code 才跟 UI 連起來，所以符合 implicitly unwrapped optional 一開始是 nil 之後會一直有值的概念。
- 一個 Storyboard UI 上的 view 物件可以**連到多個** code 裡的 IBOutlet 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 UI物件 (在此為某個UIButton) 本身會被當作參數傳入 sendClicked 函數當作sender。
- 多個 Storyboard UI 上的 view 物件可以**共用同一個** IBAction function。

六. 實驗步驟

1. 建立 iOS 專案 - Calculator

(1) 開啟Xcode，左上角menubar，選擇 File -> New -> Project

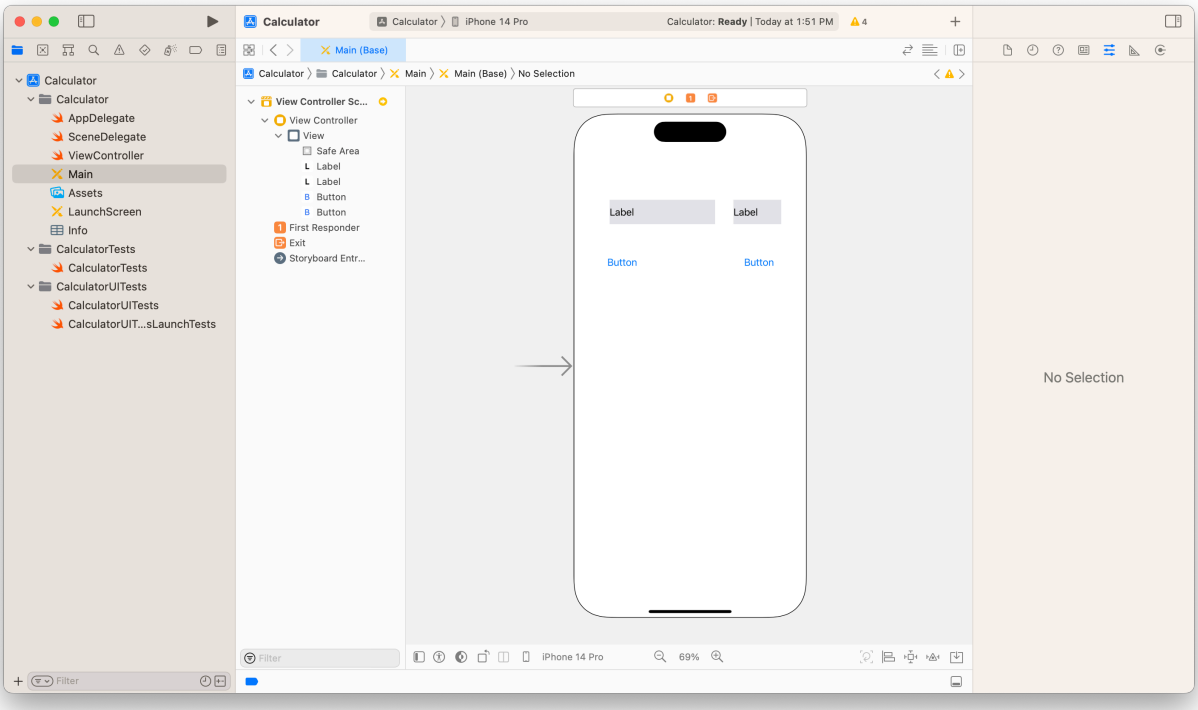
(2) 左方選擇 iOS 內的 Application ， 右方選擇 Single View Application，按 Next

(3) Product Name 輸入 **Calculator**
Organization identifier 輸入 nctu.esd.”你的學號”
Language 選擇 Swift
按下 Next，存在桌面

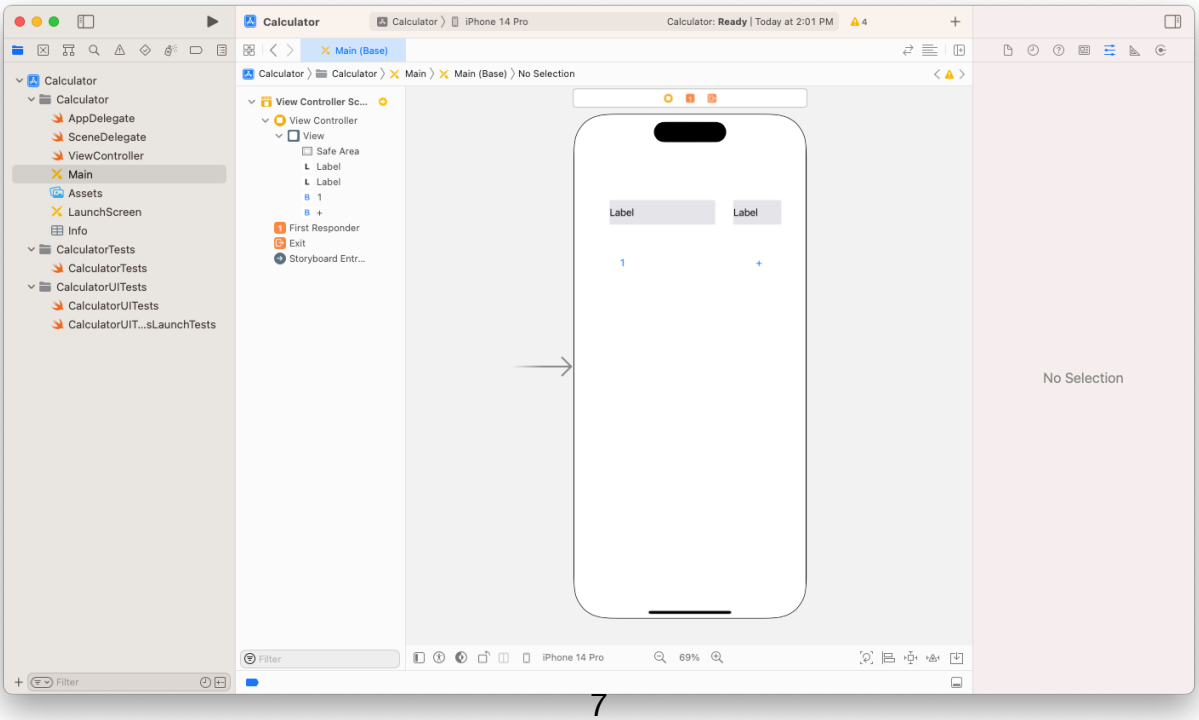
2. Calculator 基本按鈕輸入互動

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

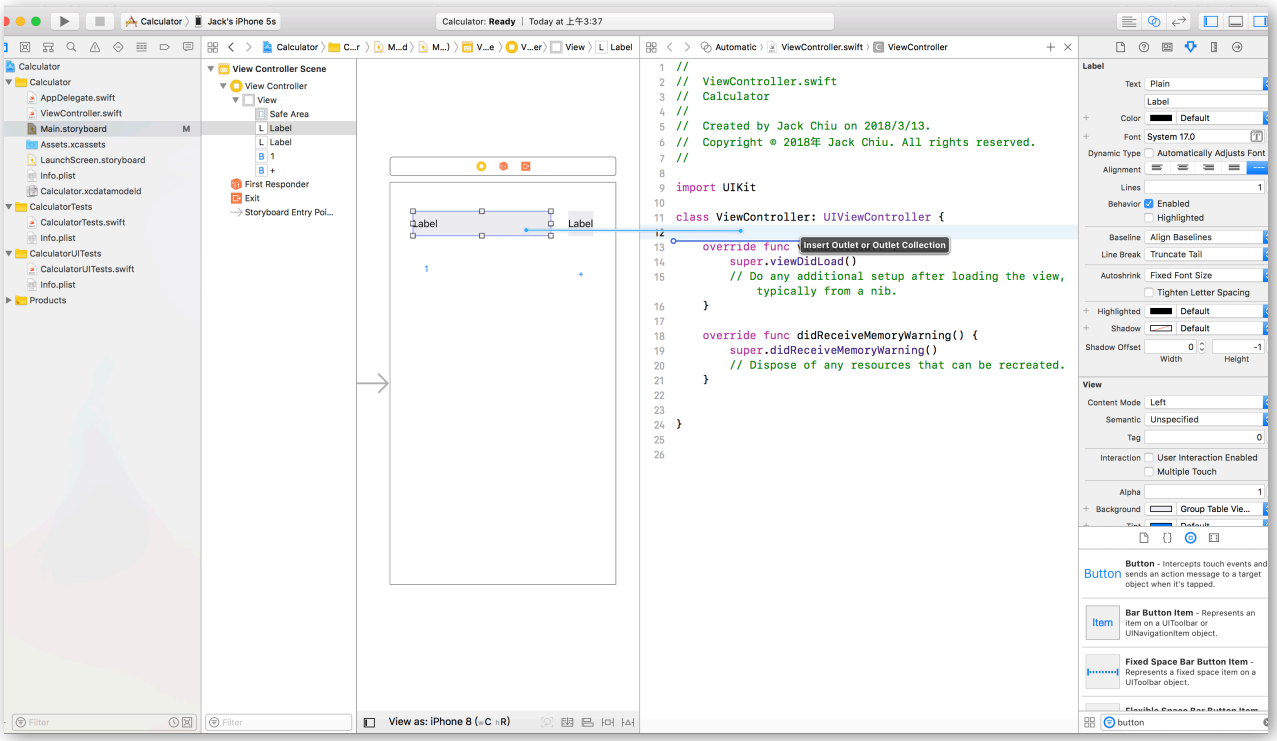
(2) 我們先從數字的輸入及運算子的輸開始著手，從右下方Show Object Library 中拉出兩個 Button (UIButton) 、 兩個 Label (UILabel) 到畫面中。



(3) 我們先以計算機介面中的數字 "1" 按鈕以及，" + "按鈕為例，寫出這兩種按鈕的基本互動。我們將按鈕文字改成" 1 "以及" + "，並將 Label 改成我們想要輸入數字以及輸入運算子的樣式。

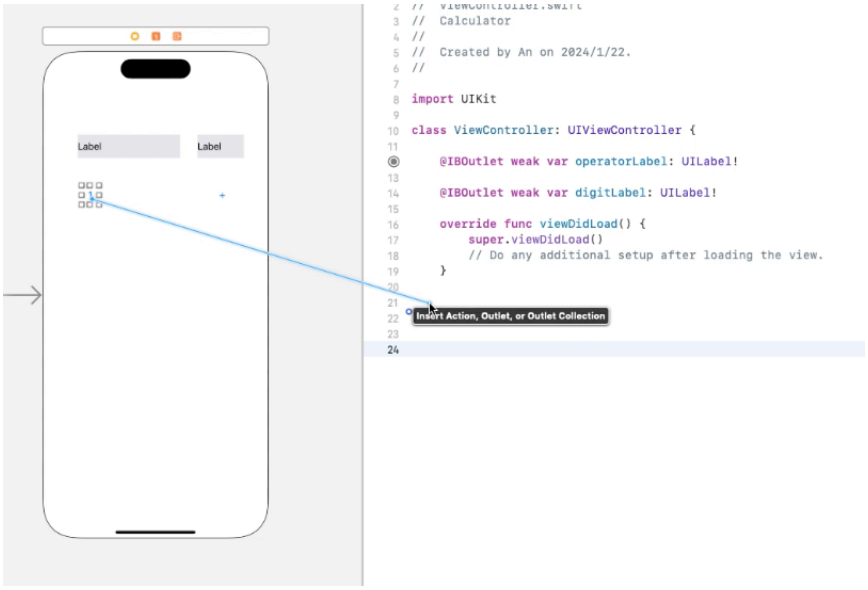


(4) 點開利用 Assistant editor 介面，利用滑鼠右鍵拖拉 Label 進入 code 介面的 ViewController Class 中，建立 digitLabel 以及 operatorLabel 兩個 IBOutlet



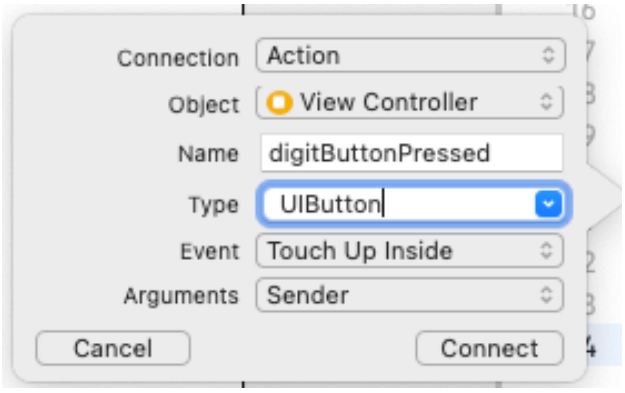

```
@IBOutlet weak var digitLabel: UILabel!
@IBOutlet weak var operatorLabel: UILabel!
```

(5) 將 " 1 " 右鍵拖入 ViewController 中，建立這個 button 對應的 method



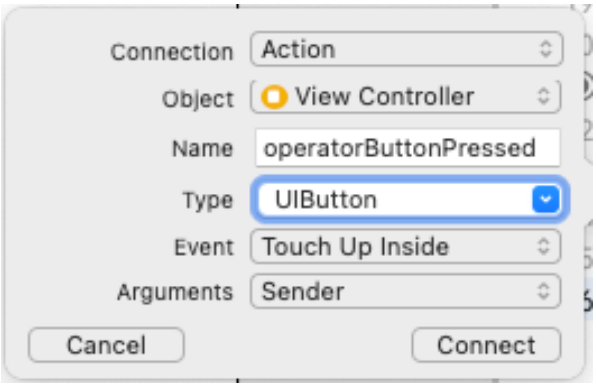
(6) 在彈出的框框中如以下設定，按下 Connect

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

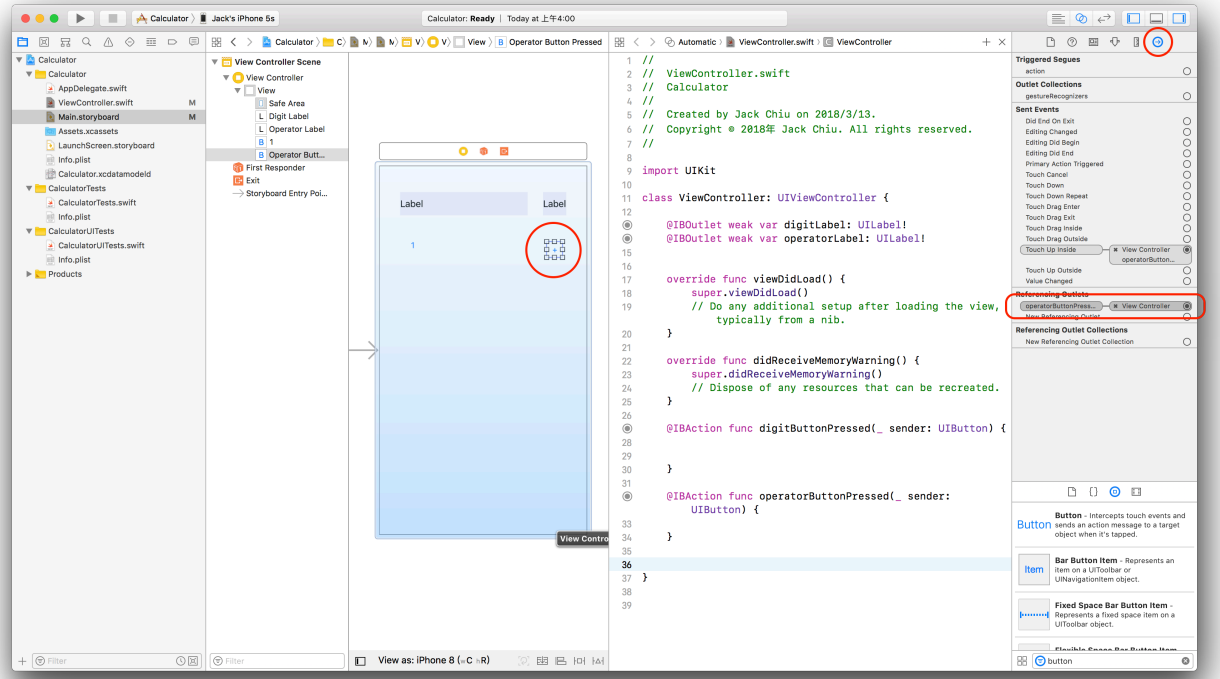


(7) 同樣將 " + "按鈕拖入 ViewController 中，建立對應的 operatorButtonPressed method，在彈出的框框中如以下設定，按下 Connect

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



※注意，如果連線的地方有做錯(例如IBAction拉成IBOutlet)，除了把錯誤的code刪掉之外，也需要去Storyboard檔案把錯誤的連線刪除，如下圖



(8) **完成按鈕的內容**。依照計算機的使用方式，在按下數字鍵時，我們希望按的數字接在原有的數字後方；而按下運算符號時，取代掉原本選取的計算符號。

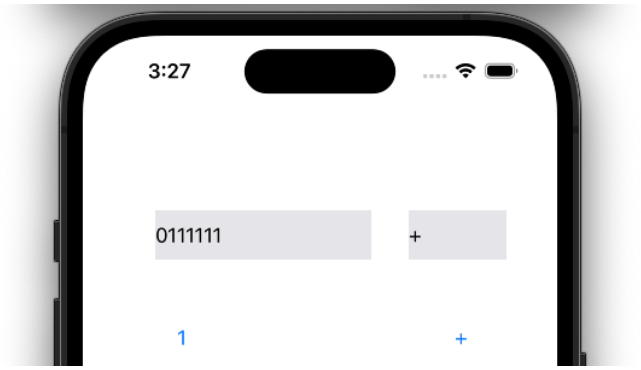
```
@IBAction func digitButtonPressed(_ sender: UIButton) {

    //將 sender button 的文字接在 digirLabel的文字後方
    digitLabel.text = digitLabel.text! + sender.titleLabel!.text!
}

@IBAction func operatorButtonPressed(_ sender: UIButton) {

    //將 sender button 的文字取代 operatorLabel 原有的文字
    self.operatorLabel.text = sender.titleLabel?.text
}
```

※ sender 就是 trigger 這個 method 的物件，我們這次實驗直接將其型態設定成 UIButton，就可以直接調用 sender 的 titleLabel

(9) **設定digitLabel及operatorLabel的初始狀態**。在計算機初始時，我們希望數字欄位顯示 " 0 "，而運算符號是空的。我們可在 viewDidLoad 設定 label 的初始狀態：

```
override func viewDidLoad() {
    super.viewDidLoad()

    //初始化數字及運算子的狀態
    digitLabel.text = "0"
    operatorLabel.text = " "
}
```

(10) **執行App**。測試看看兩個按鈕的動作是否如我們所願



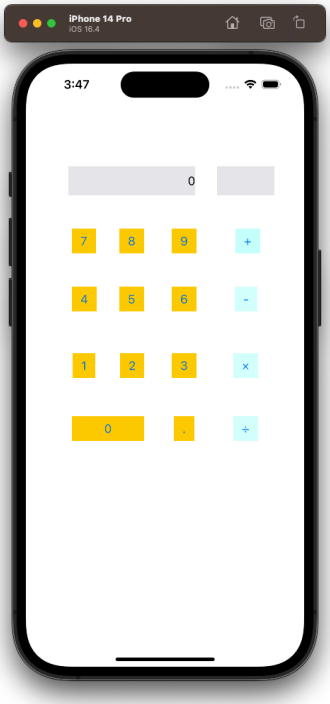
(11) 由執行結果可發現，digitLabel初始0的狀況需要特別處理，不然會出現0在數字前面的狀況，因此我們稍微修改一下程式碼：

```
@IBAction func digitButtonPressed(_ sender: UIButton) {
    //如果按下按鈕時，digitLabel為初始0的狀態，把初始的0刪掉後再開始輸入
    if digitLabel.text == "0"{
        digitLabel.text = ""
    }
    //將 sender button 的文字接在 digirLabel的文字後方
    digitLabel.text = digitLabel.text! + sender.titleLabel!.text!
}
```

(12) 可自行執行看看修改後的結果

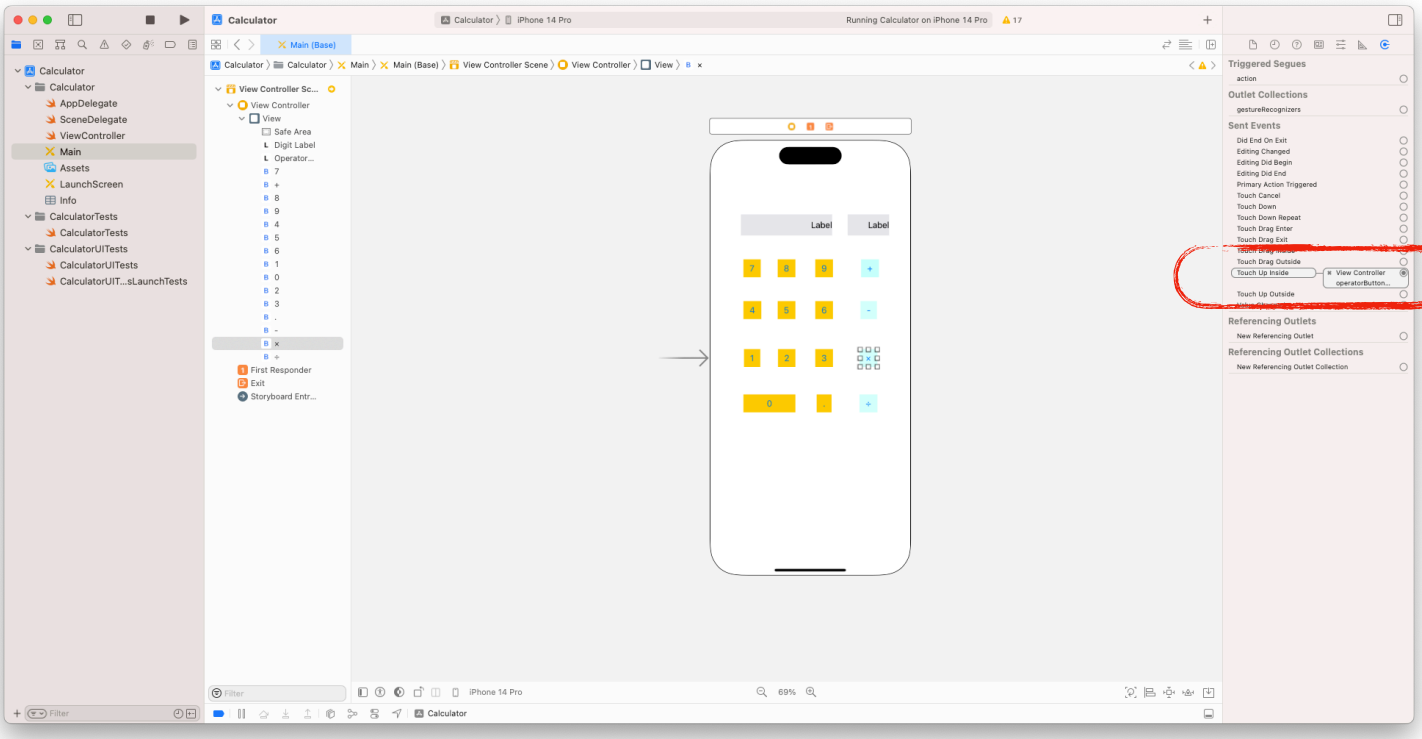
3. 完成 Calculator 的介面

(1) 依照一般簡單計算機的介面，我們需要 1 ~ 9 的數字按鈕， " . " 的按鈕以及 "+ − × ÷ ="的按鈕。由於這些按鈕按下後的的行為幾乎都與我們寫好的 digitButtonPressed 和 operatorButtonPressed 相同，我們可以直接複製原本已經在界面上的按鈕，會連已經跟 code 連接好的 method 一起複製。需要新拉進來的只有 " = " 的按鈕以及清空用的 " AC " 按鈕

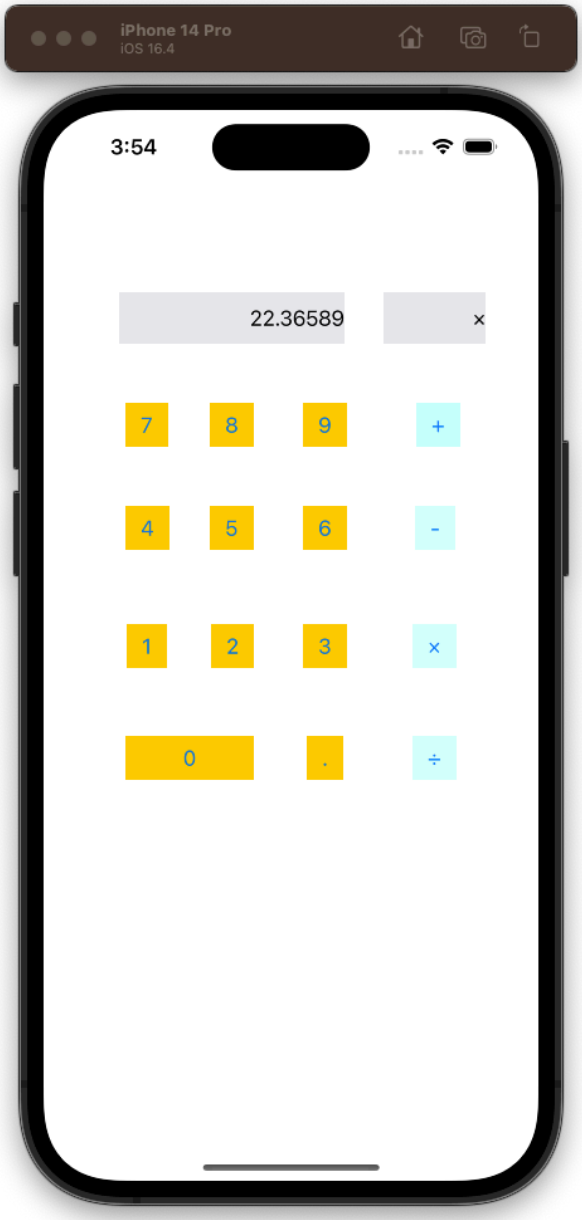


※排版不一定要跟講義一樣，可以自由發揮創意來設計更好看的版面喔>w<

(2) 檢查一下是否每個數字按鈕都有連接 digitButtonPressed ，所有運算子按鈕都有連接operatorButtonPressed



(3) 此時執行，可發現所有數字按鈕及運算子按鈕都已可以運作



4. 完成 Calculator 輸入行為

(1) **小數點輸入的處理。**此時數字輸入剩下一個問題，就是在輸入" . "的時候，如果前面是0，0會被取代掉，而且，一個數字中也可以重複出現兩個 " . "，在此，我們把這兩個例外考慮進 digitButtonPressed 中

```
@IBAction func digitButtonPressed(_ sender: UIButton) {
    //如果按下按鈕時，digitLabel為初始0的狀態，把初始的0刪掉後再開始輸入
    //如果是按下"."，不用刪掉0
    if digitLabel.text == "0" && sender.titleLabel?.text != "."
    {
        digitLabel.text = ""
    }
    //為了不讓一個數字出現兩個"."，如果按下"."且原本的數字已經有"."，則跳出
    if sender.titleLabel?.text == "." &&
        digitLabel.text?.range(of: ".") != nil
    {
        return
    }
```

(2) **判斷開始輸入第二個運算數字。**當我們按下operator按鈕後，下一個輸入的阿拉伯數字數字應該是第個運算數字的開頭，而不是繼續接在前一個數字後面，而此時顯示在螢幕上的第一個輸入的數字也應該存起來，作為最後計算時的第一個運算數字，在此，我們可在 ViewController class 中加入兩個 property (member paramerter)來達成：

```
//表示下一次按digit按鈕時要開始輸入一個新的數字
var shouldStartNewNumberInput = false

//按下operator前輸入的數字暫存在這裡
var pendendingNumber = ""
```

在 operatorButtonPressed method 中將 shouldStartNewNumberInput 改成 true

```
@IBAction func operatorButtonPressed(_ sender: UIButton) {
    //將 sender button 的文字取代 operatorLabel 原有的文字
    self.operatorLabel.text = sender.titleLabel?.text

    //已按下運算子，下一個digit輸入時應該開始新的數字輸入
    shouldStartNewNumberInput = true
}
```

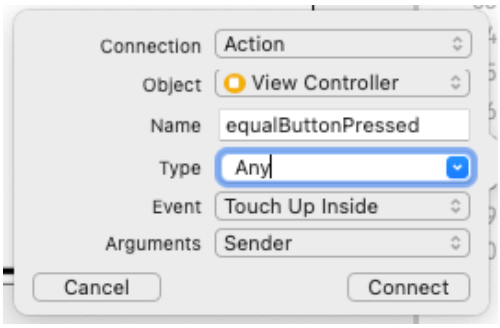
在 digitButtonPressed method 一開始時，新增 shouldStartNewNumberInput == true 時的行為

```
@IBAction func digitButtonPressed(_ sender: UIButton) {
    //判斷是否開始新的數字輸入
    if shouldStartNewNumberInput{
          //暫存前一個輸入的數字
          pendendingNumber = digitLabel.text!

          //初始化數字輸入匡，初始值是0，與viewDidLoad一樣
          //(剛按下的新的digit在if過後會放到digitLabel)
          digitLabel.text = "0"

          //開始新的數字輸入了，把flag改回來
          shouldStartNewNumberInput = false
       }
    if digitLabel.text == "0" && sender.titleLabel?.text != "."{
          digitLabel.text = ""
    }
    if sender.titleLabel?.text == "." &&
          digitLabel.text?.range(of: ".") != nil
          {
            return
          }
    digitLabel.text = digitLabel.text! + sender.titleLabel!.text!
  }
```

5. 完成 Calculator 計算行為

(1) 由於" = "按鈕還沒有對應的 method，所以我們再以右鍵拖移的方式建立" = "按鈕的 IBAction method，在彈出的框框中如以下設定，按下 Connect

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



※這裡的 Type 不用特別選 Button ，因為不需要跟sender取值來用

(2) 如果什麼運算子 (+ − × ÷)都還沒按，" = "按鈕就不應有動作，此時我們可以利用Lab 1提到的 guard let 來達成

```
    @IBAction func equalButtonPressed(_ sender: Any) {
        //檢查operatorLabel有沒有值，如果是nil或是空字串，則離開function
        guard let operatorString = operatorLabel.text,
                    !operatorString.isEmpty
        else { return}
    }
```

(3) 確定運算子存在後，我們需要取得要被運算的兩個數字，由於我們前面都是使用string來存輸入的數字，此時我們須將string轉換成可以計算用的Double。這邊要注意的是，由於不是所有string都可以轉成Double，所以，轉型完後的型態會是optional Double，需要在unwrap 過後才能當作Double來做加減乘除

```
    @IBAction func equalButtonPressed(_ sender: Any) {
        //檢查operatorLabel有沒有值，如果是nil或是空字串，則離開function
        guard let operatorString = operatorLabel.text,!
operatorString.isEmpty
        else { return}

        //檢查 pendingNumber 與 digitLabel 的字串可否轉成數字，若不行則離開
        //若可以轉成數字，unwrap成Double存到value1, value2
        guard let value1 = Double(pendendingNumber),
              let value2 = Double(digitLabel.text!) else { return}

     }
```

(4) 完成四則運算計算，並且顯示結果在螢幕上，請自行將剩餘的+ − × ÷運算子case 補齊

```
@IBAction func equalButtonPressed(_ sender: Any) {

    guard let operatorString = operatorLabel.text,
              !operatorString.isEmpty else { return}

    guard let value1 = Double(pendendingNumber),
          let value2 = Double(digitLabel.text!) else { return}
    //暫存計算結果的變數
    var result:Double = 0
    //根據不同的 operator 做計算
    switch operatorString {
    case "+":
        result = value1 + value2
    // ... add your own case
    default:
        break;
    }
    //將計算的結果顯示在 digitLabel
    digitLabel.text = "\(result)"
}
```

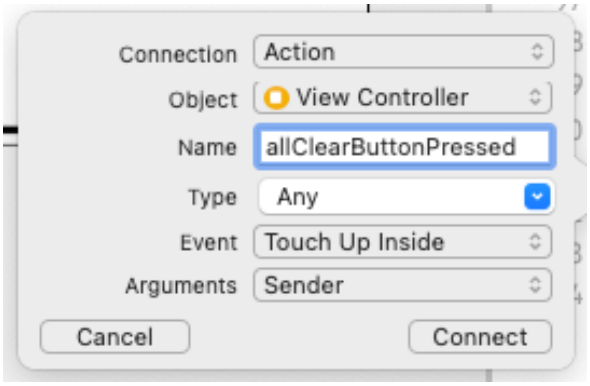
(5) 最後，依照一般計算機的使用方式，將計算機改為可以輸入下一次計算的狀態，在上面的程式後接續加上以下程式

```
    //將螢幕上的運算子清空
    operatorLabel.text = ""

    //按下等號後，下一次按Digit為輸入一個新的數字(第一個運算數字)
    shouldStartNewNumberInput = true
```

(6) 可自行執行看看執行後的結果，+ – × ÷ 的結果是否都正確

6. 完成 Calculator 計算行為

(1) **我們剩下 "AC" (All Clear)按鈕的動作要完成。**先將"AC"按鈕的 IBAction method建立

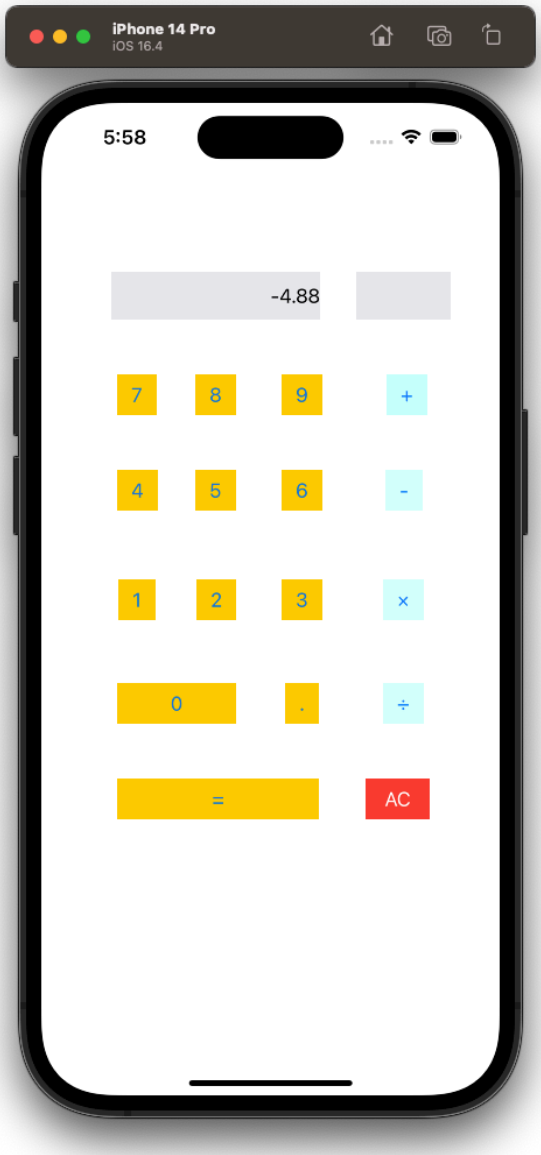


(2) 按下"AC"後，計算機要回復到初始的狀態，為了避免遺漏，我們將現有的變數（包含連接 UI 的 outlet）：digitLabel、operatorLabel、shouldStartNewNumberInput、pendendingNumber都回復到初始狀態

```
@IBAction func allClearButtonPressed(_ sender: Any)    {
    digitLabel.text = "0"
    operatorLabel.text = ""
    shouldStartNewNumberInput = false
    pendendingNumber = ""
}
```

7. 執行結果

(1) 在計算機輸依序輸入 "1.2 × 0.1 = - 5 = " 的結果



七. Demo

1. 請 Demo 基本計算機最後的執行結果，**加減乘除、小數點需可以正常計算且AC可以使用**，排版不用跟講義一樣

2. 加分題

   (1)**基礎加分題：**加入 back 鍵

   例： digitLabel 上顯示 "123" ，按下 back 後顯示為 "12"，不影響其他行為，如選擇operator或前一次計算。

   (2)**進階加分題：**連續輸入(照順序，不用先乘除後加減)

   例： 依序輸入 "1 + 2 + 3 × 2 = "，結果為 12

   (3)**進階加分題：**連續計算

   例： 依序輸入 "1 × 2 = = = ="，結果為 16

※基礎加分題做完才能做進階加分題

※進階加分題不分順序，擇一做就有加，兩個都做加最多

※超過下課時間後就不能Demo加分題