Swift App 實作入門 for iPad

教師專班 Day 1

主辦：新北市政府教育局
日期：2023.03.11 (六)
講師：友教有限公司 Michael

Playgrounds 軟體版本 1

從積木程式到文字程式語言 3

SwiftUI 體驗 5

畫面排版 – VStack, HStack 9
畫面互動 – 按鈕事件與圖片 13
文字互動 – TextField、Button、Text 21

條件碼 24

終極密碼 APP 25

小組討論 25
遊戲畫面規劃 27
遊戲邏輯判斷 –– 按鈕與條件碼 28
更完整的遊戲設計 29

猜燈謎 APP 32

陣列資料 Array 32
加入畫面元件 32
遊戲邏輯判斷 34
定義 struct 結構 35

Playgrounds 軟體版本

版本需求 4.1 以上，目前已更新至版本 4.2。

iPad：

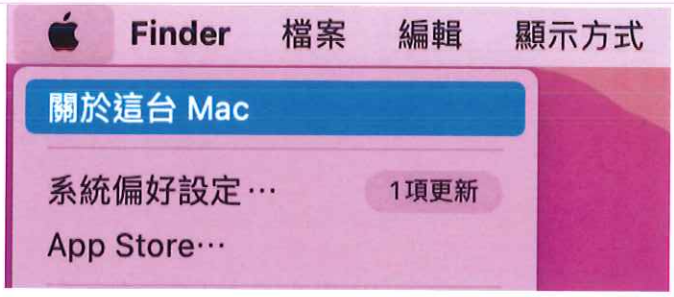
進入 ipad 的設定中找到『一般』

選擇『關於本機』，可以找到 ipadOS 的版本

在設定中往下尋找『Playgrounds』，點選進入後可以找到 Playgrounds 的版本。

Mac：

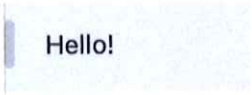
Mac 左上角點選蘋果圖示，找到『關於這台 Mac』，查看macOS版本 。



打開 Playgrounds 後，在左上角點選『 關於 Playgrounds 』，查看軟體版本。

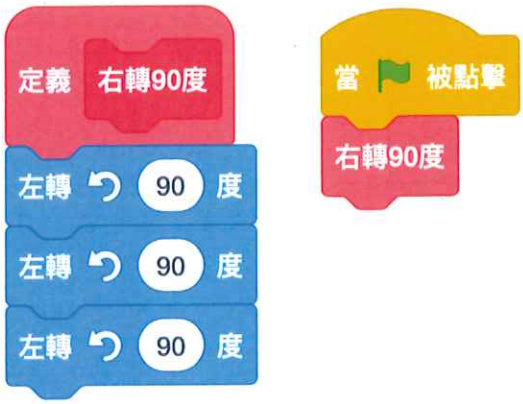
從積木程式到文字程式語言

指令

Scratch積木程式	Swift 文字程式碼
	`show("Hello!")`
顯示區	顯示區
	

函數

Scratch積木程式	Swift 文字程式碼

	`func turnright(){`
	`  turnLeft()`
	`  turnLeft()`
	`  turnLeft()`
	`}`
	
	`turnright()`

For 迴圈

Scratch積木程式	Swift 文字程式碼

	`for i in 1 ... 3 {`
	`  turnLeft()`
	`}`

條件碼

1. 設定『當綠旗子被按下後，角色會詢問9X3=?』
2. 使用如果–否則積木，設定條件內容為『詢問的答案 = 27』
3. 若上方條件成立，角色說出答對了，否則說出答錯了。

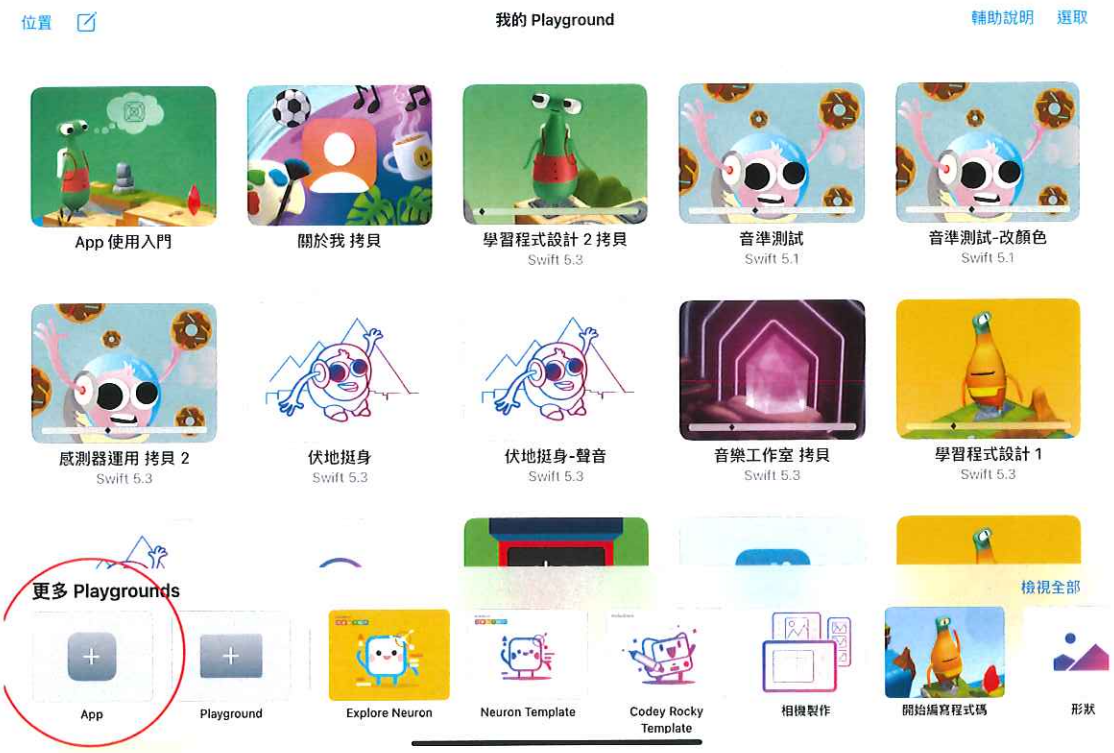
Scratch積木程式	Swift 文字程式碼

	`show("9x3 = ? ")`
	`var ans = askForNumber("答案是？")`
	
	`if ans == 27 {`
	`  show("答對了")`
	`} else {`
	`  show("答錯了")`
	`}`

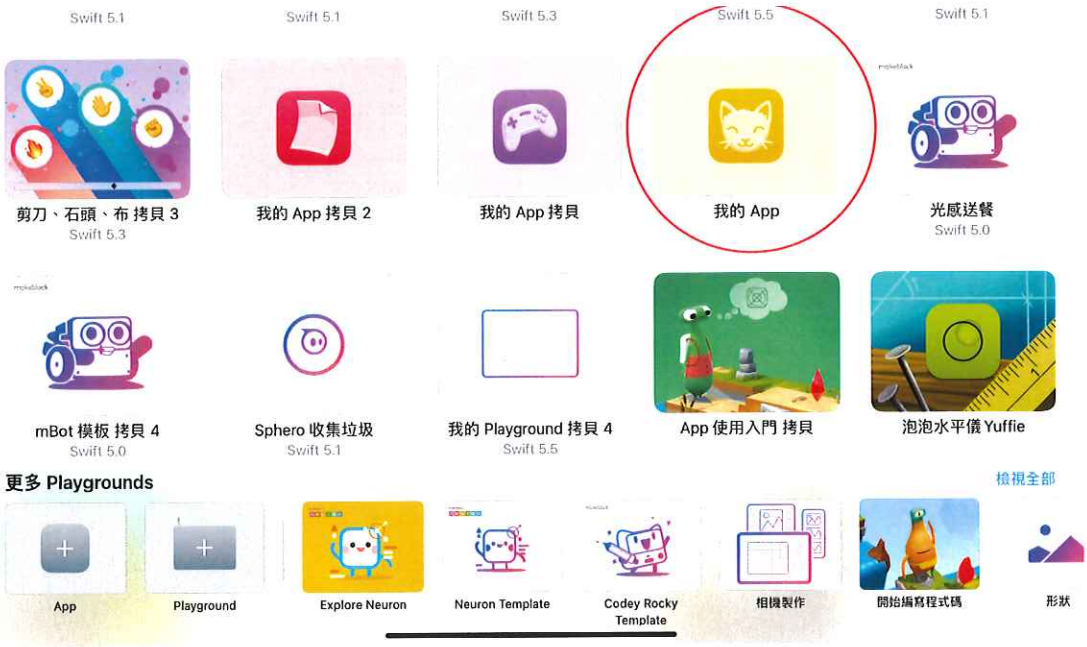
SwiftUI 體驗

新增一個 APP

在首頁按下，下方更多 Playgrounds 的 App + 。如下。

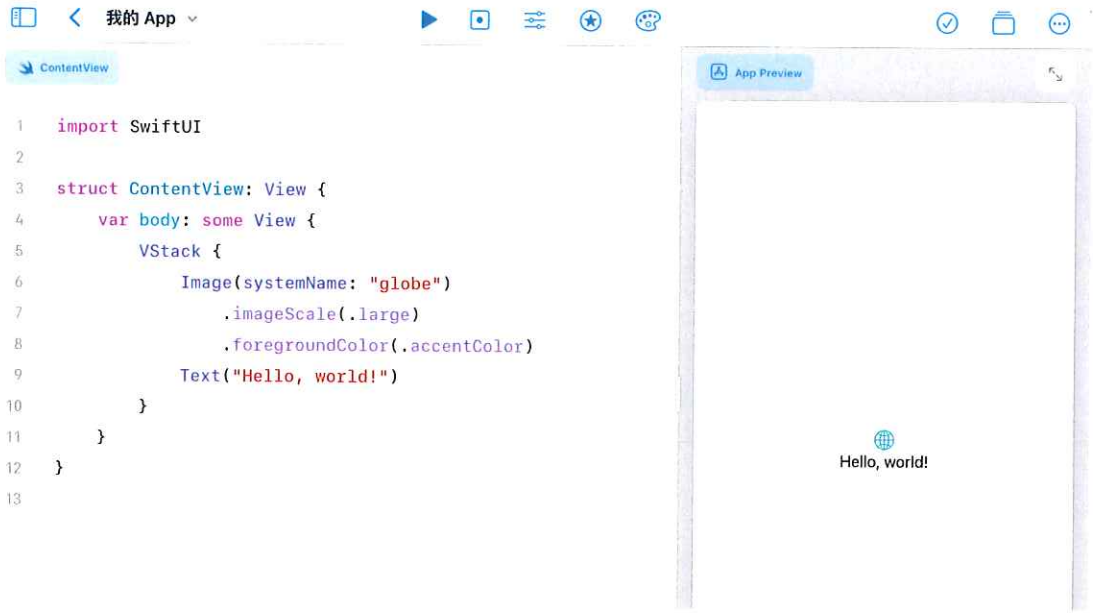


會看到多一個 App 專案，如下圖。App 圖片會隨機給一個，名稱會叫 **我**的 App or **我**的 App **拷貝** (**數**字)。

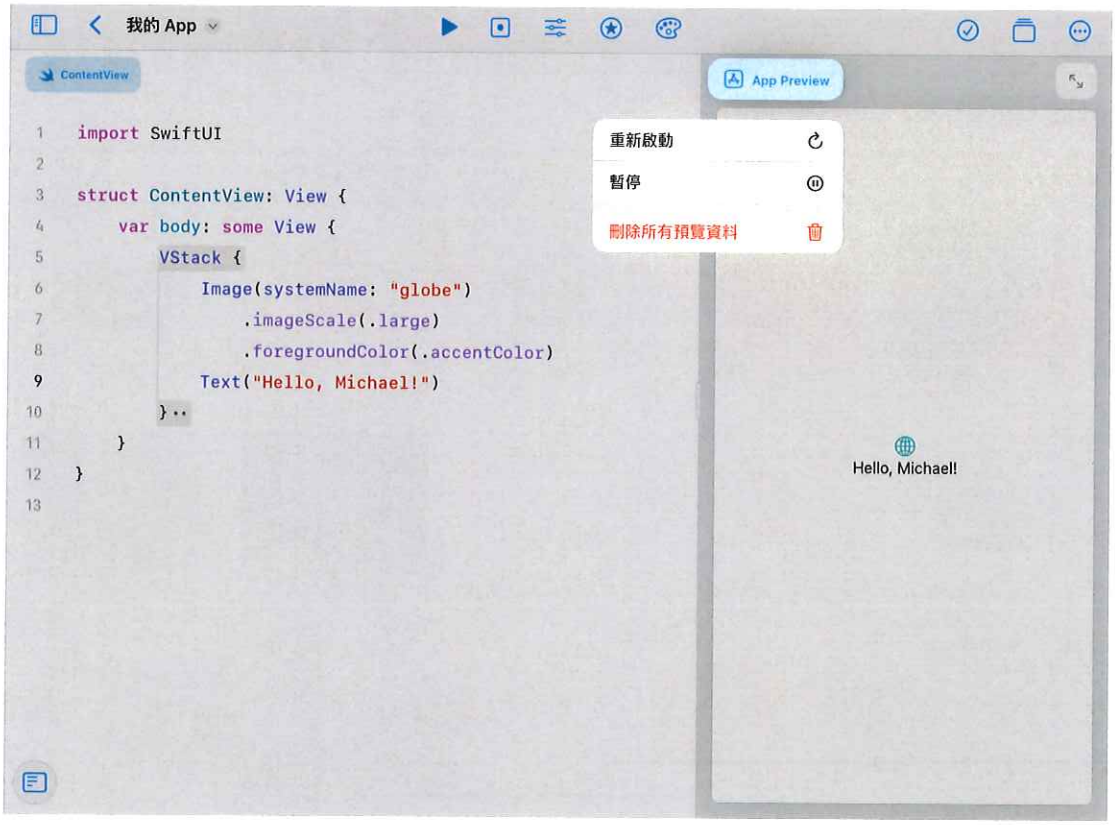


打開**我**的 App 後左邊是寫程式的地方，右邊是左邊程式的結果。只是右邊沒有執行**我**的程式的按鈕。

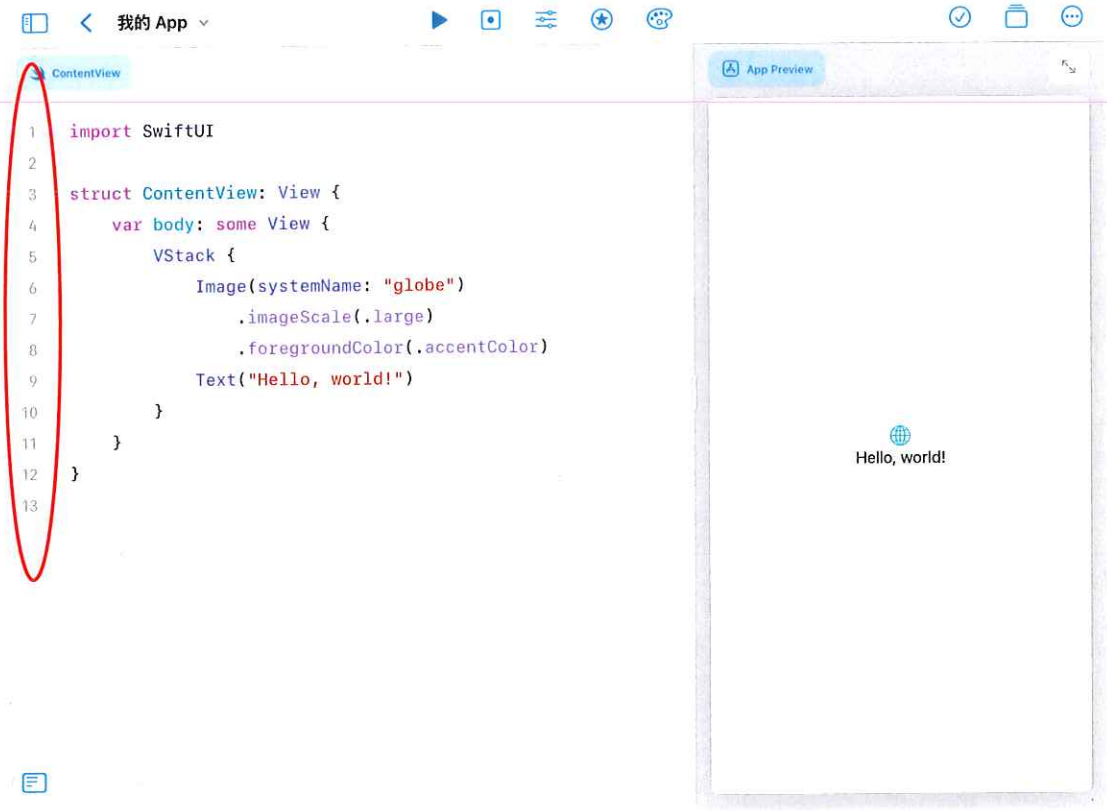
原則上系統會自動執行，所以不用手按，改變左邊內容右邊就會改變。



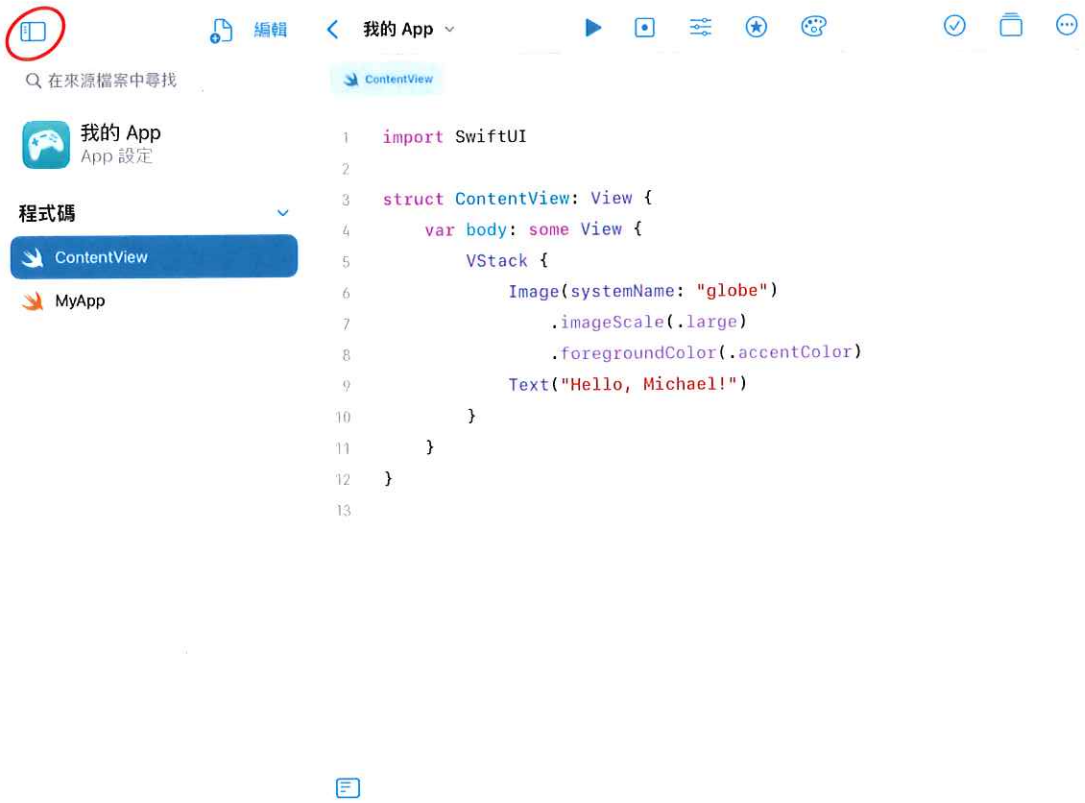
把 world 改成 Michael ，右邊會自動執行。如果沒有自動執行，可以按下右上的 App **預覽**，就有 Restart Preview 可以點選。



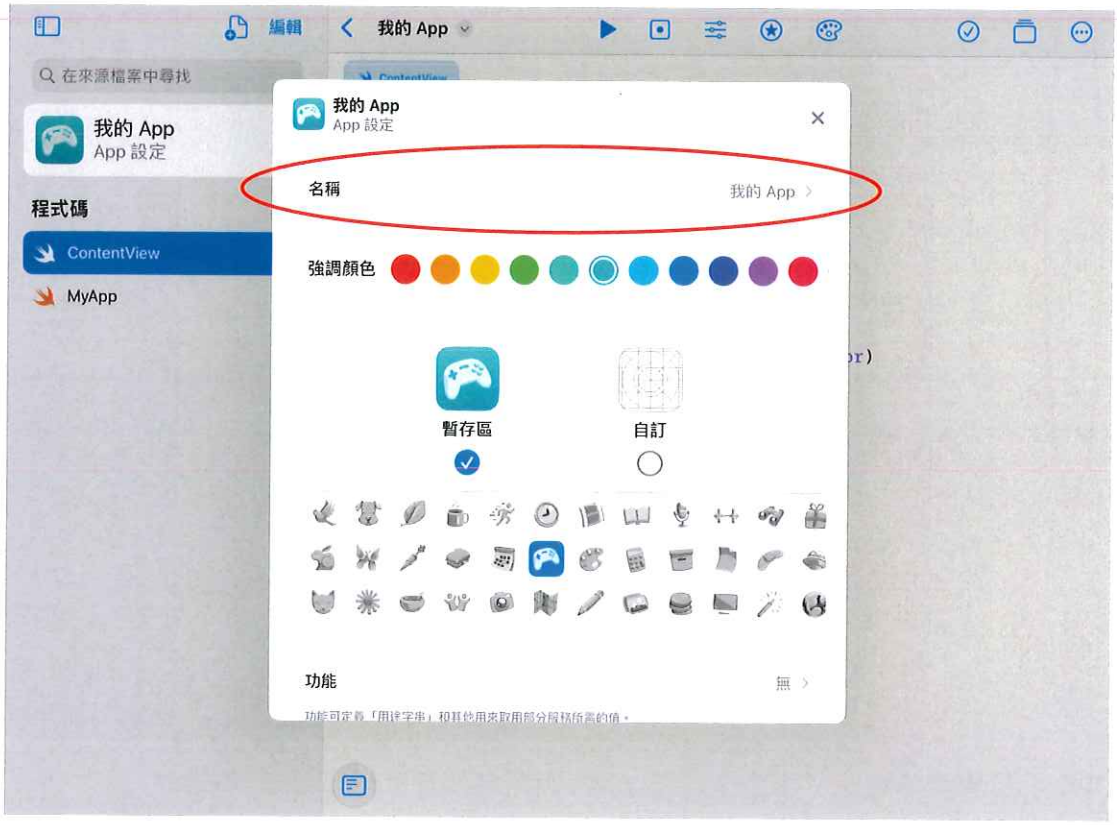
每一列程式都有 Line number ，用來方便教學時候特別指出在講解哪段程式碼。



按一下左上角的這個圖示，會顯示這個 APP 內所有的程式碼檔案與資料。



點選**我的** App 設定會跳出視窗可以改名稱，如下。這個名稱會在上架審核用到，和 Swift Playgrounds 顯示名稱不同步。

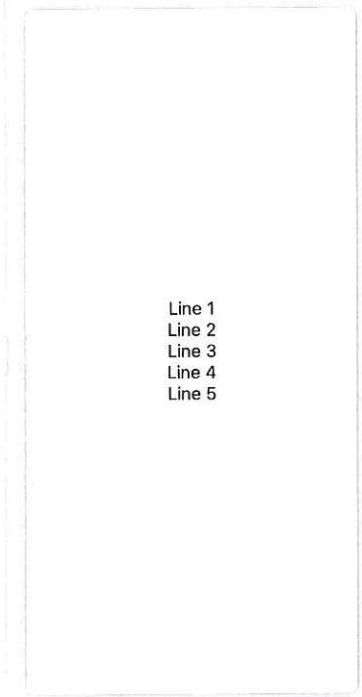


畫面排版 – VStack, HStack

SwiftUI 的畫面和程式寫出來的順序有關係，而畫面的排版就要利用 VStack 垂直排列 or HStack 水平排列來完成。而眼睛看得到的畫面是**預**先定義好的類型名稱，也可以想像成用文字來表示的積木方塊，比如 Text("Hello")。Text 就是文字畫面的類型名稱，() 裡面的 "Hello" 是文字內容。排列一個垂直的一串文字如下表示。

```
struct ContentView: View {
    var body: some View {
        VStack {
            Text("Line 1")
            Text("Line 2")
            Text("Line 3")
            Text("Line 4")
            Text("Line 5")
        }
    }
}
```

會在右邊畫面**預覽**看到。

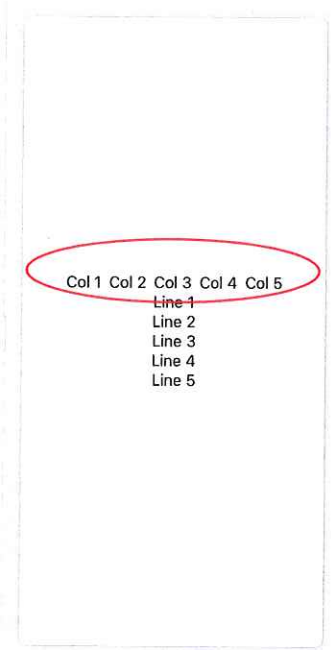


SwiftUI 的排列原則就是，向中心對齊。

接著試試 VStack ， 結合 HStack。

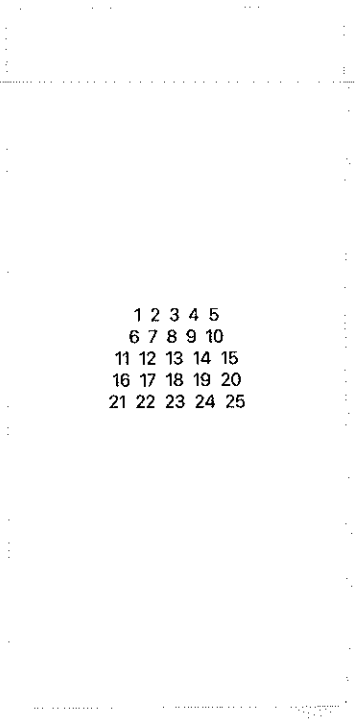
```
struct ContentView: View {
    var body: some View {
        VStack {
            HStack{
                Text("Col 1")
                Text("Col 2")
                Text("Col 3")
                Text("Col 4")
                Text("Col 5")
            }
            Text("Line 1")
            Text("Line 2")
            Text("Line 3")
            Text("Line 4")
            Text("Line 5")
        }
    }
}
```

得到的畫面如下。



HStack 會把 { } 裡面的畫面元件都水平排列，並且佔用一個 Line 的空間。

Challenge：請完成如下畫面 – 30 mins



預計用到類型：

實際用到類型：

章節筆記

語法疑問：

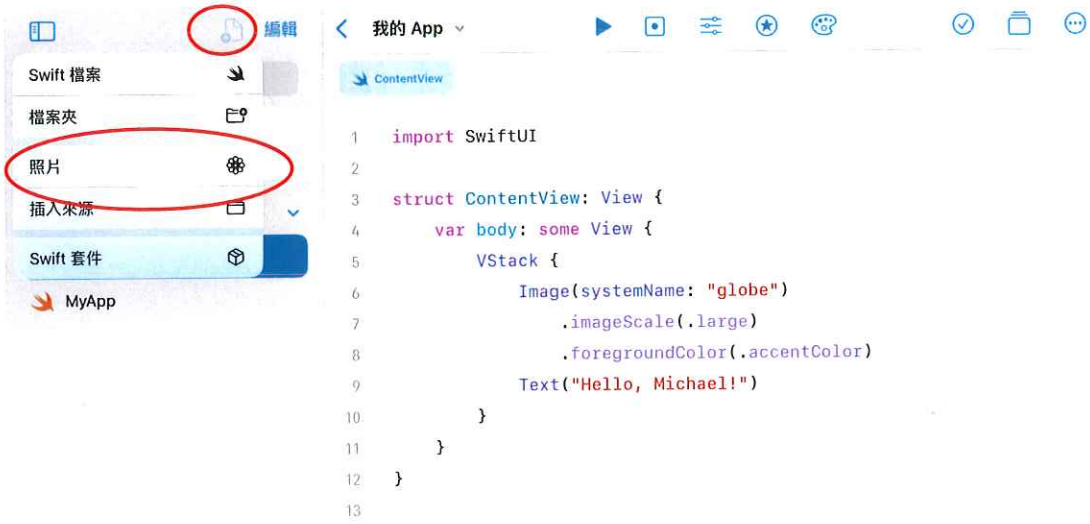
教學重點：

反思回饋：

畫面互動 – 按鈕事件與圖片

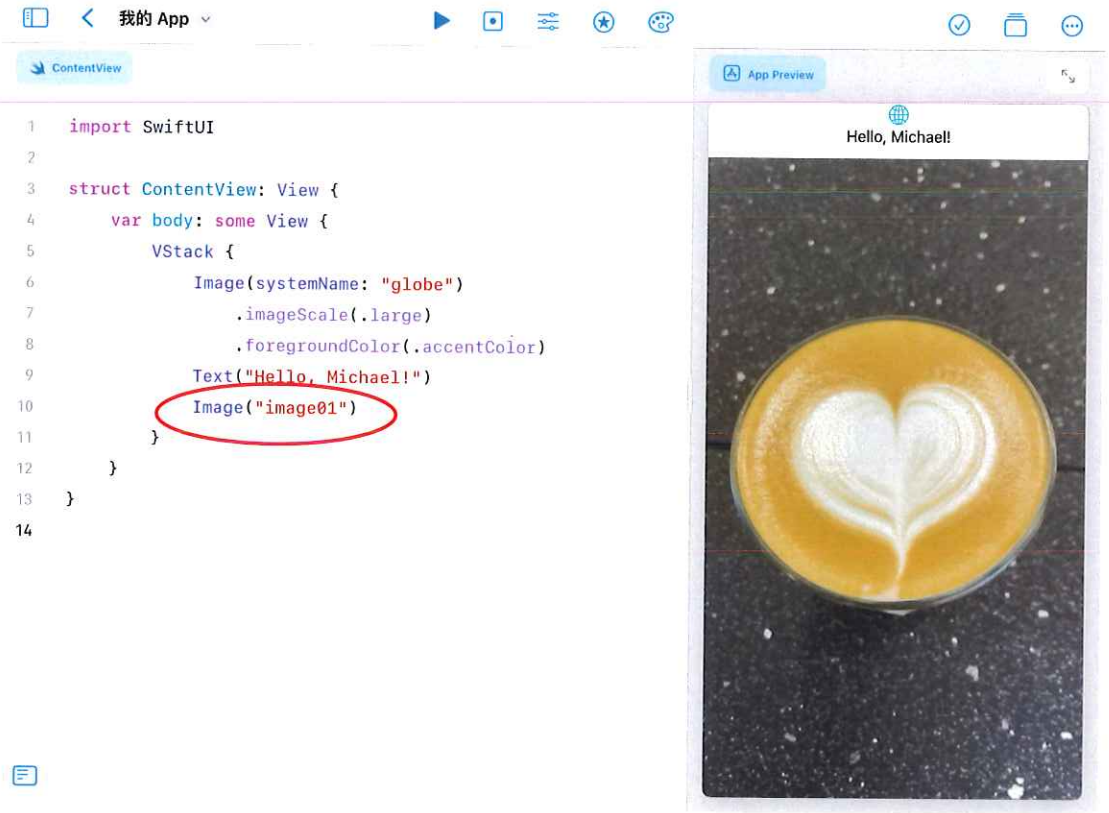
新增自定義圖片

按下右上方新增，選擇照片。



點選想要放入的圖片後，圖片會放在內容底下，長按後出現重新命名的選項，幫圖片重新命名。並在 body 裡面加入一張圖片，程式碼如下。

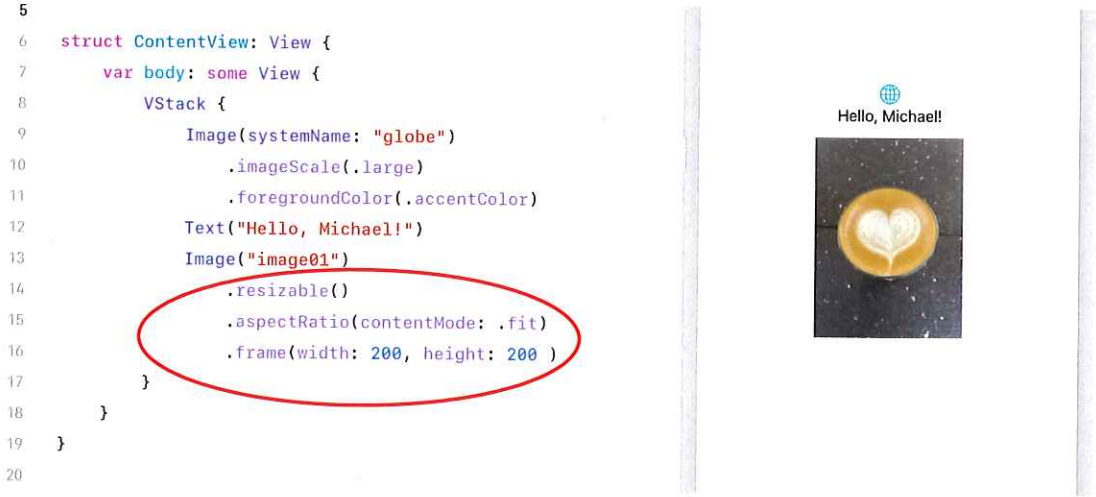
打開右邊預覽畫面，看圖片是否有正常顯示在“Hello, World!”的下方。




```
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {
            Image(systemName: "globe")
                .imageScale(.large)
                .foreground Color(.accentColor)
            Text("Hello, Michael!")
            Image("image01")
        }
    }
}
```

接著試試看以下的程式碼來調整圖片大小的設定。




```
struct ContentView: View {
    var body: some View {
        VStack {
            Image(systemName: "globe")
                .imageScale(.large)
                .foreground Color(.accentColor)
            Text("Hello, Michael!")
            Image("image01")
                .resizable()
                .aspectRatio(contentMode: .fit)
                .frame(width: 200, height: 200 )
        }
    }
}
```

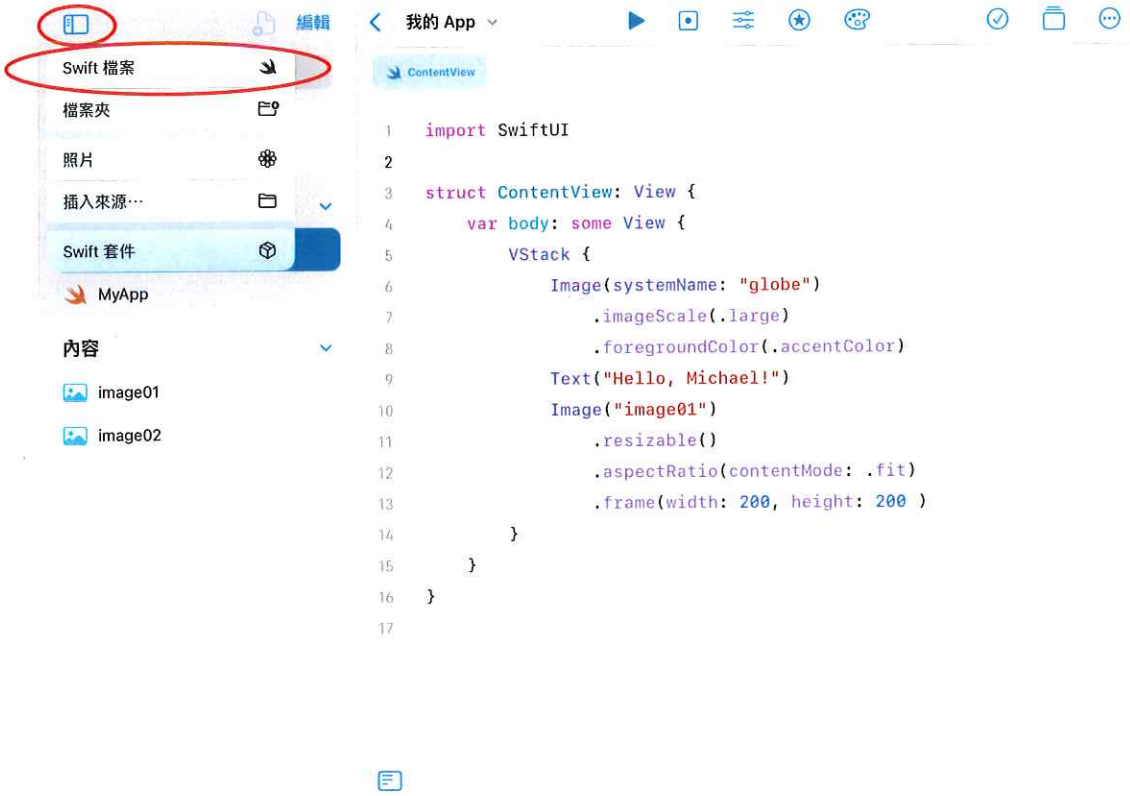
Thinking Time ：為什麼在 .frame(width: 200, height: 200) 裡面所設定的大小為正方形，但顯示的結果不是呢？

Thinking Time ：點 Dot (.)的用法是什麼？

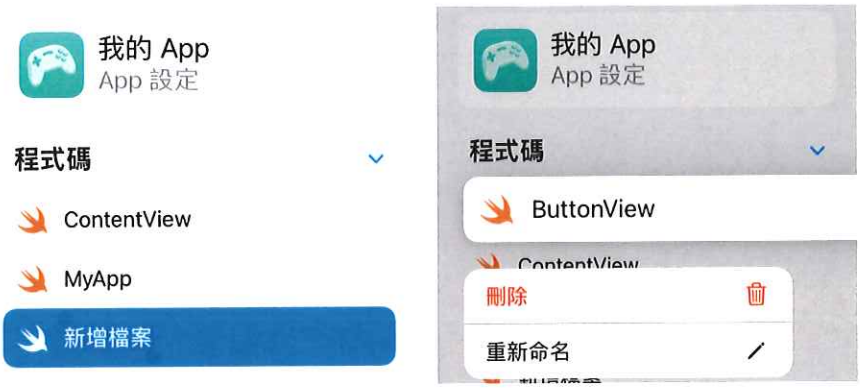
新增按鈕

在畫面中添加一個按鈕 button 元件，當按下按鈕後，換成另一張圖片的內容。

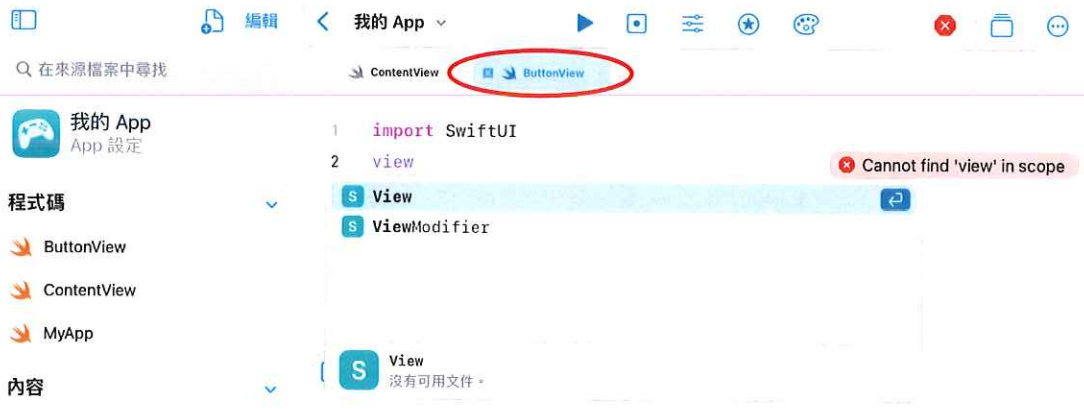
首先新增一個新的 Swift 檔案。



新增後檔案會出現在程式碼的分類底下，此時可以直接重新命名這個 Swift File（左圖），或是長按要改變檔名的檔案，會跳出可以重新命名的選單（右圖）。



在 ButtonView 檔案內，輸入關鍵字 view ，選第一個。



點選第一個 View 結構後，會自動插入一段程式碼 (左圖) ，在 MyView 的地方重新命名，更改這個新增的畫面名稱後，下方兩個藍色方框內將自動被更新 (右圖)。

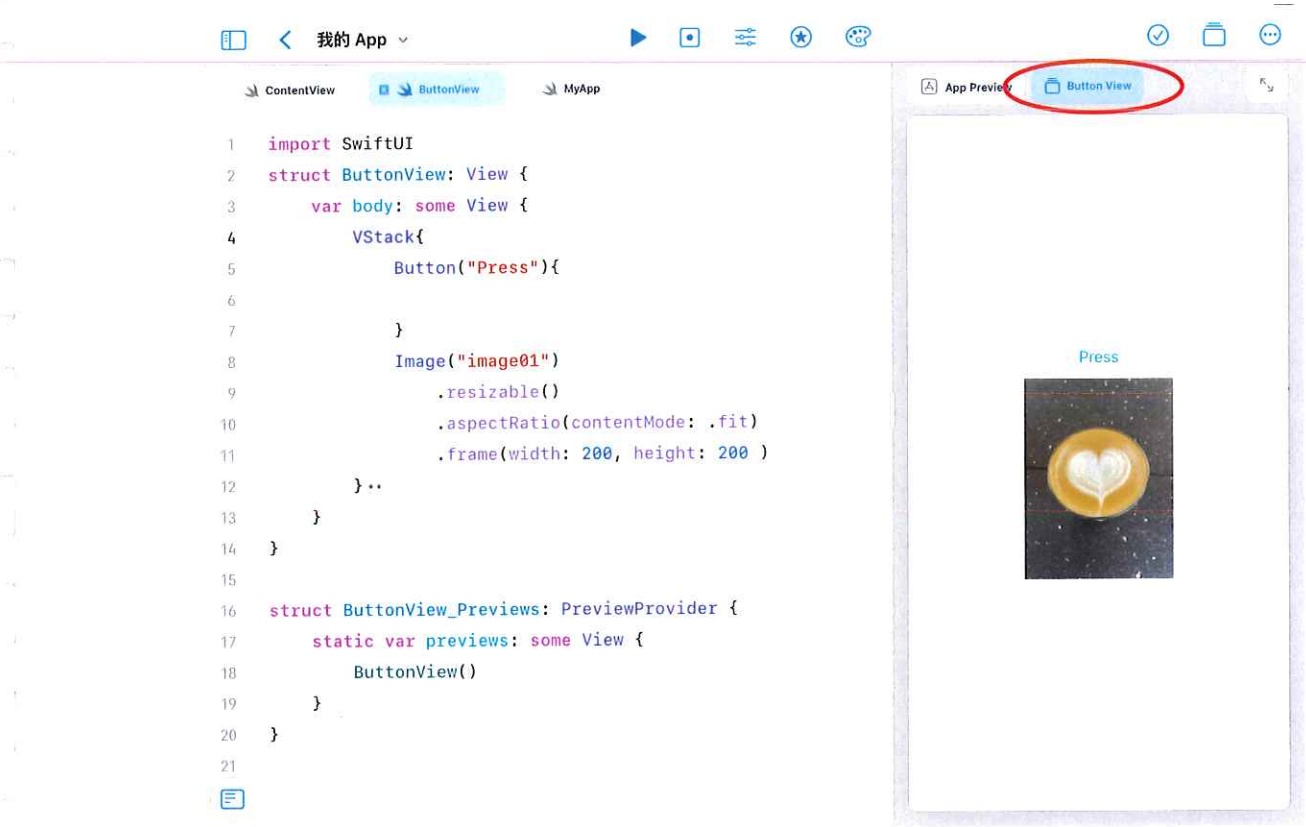
```
1  import SwiftUI
2
3  struct MyView: View {
4      var body: some View {
5          Hello, world!
6      }
7  }
8
9  struct MyView_Previews: PreviewProvider {
10     static var previews: some View {
11         MyView()
12     }
13 }
14
```

```
1  import SwiftUI
2
3  struct ButtonView: View {
4      var body: some View {
5          Hello, world!
6      }
7  }
8
9  struct ButtonView_Previews: PreviewProvider {
10     static var previews: some View {
11         ButtonView()
12     }
13 }
14
```

在 body 中加入程式碼，使用上方的元件庫來新增一個 VStack 與 一個按鈕 Button。

```
1  import SwiftUI
2
3  struct ButtonView: View {
4      var body: some View {
5          VStack{
6              Button("press"){
7
8              }
9          }
10     }
11 }
```

再新增一個圖片 Image，並打開預覽畫面，點選上方的頁面切換，往右按切換到這個檔案的預覽畫面。



若要改變預覽首頁畫面，執行程式後的畫面，要從 myApp 的檔案裡更改，將原本顯示 ContentView 的畫面改成顯示 BottonView 。

加入按鈕事件改變圖片

接下來要設定當按鈕按下去時，把圖片改成另一張圖片，此時需要使用變數的觀念，將圖片的名稱置換成變數。在這邊要注意宣告變數的位置，body 內是畫面相關內容，宣告變數 imageName 要放在 body 的結構外面。

```
3   struct ButtonView: View {
4       var imageName = "image01"
5       var body: some View {
6           VStack{
7               Button("Press"){
8
9               }
10              Image(imageName)
11                  .resizable()
12                  .aspectRatio(contentMode: .fit)
13                  .frame(width: 200)
14          }
15      }
16  }
```

將圖片名稱置換成變數後，imageName 沒有改變，畫面仍然顯示 image01 這張圖片，按鈕 Button目前是沒用途的。

接下來要設定按下 Button 後，透過改變 imageName 的內容，改變 Image 的內容來顯示另外一張圖片。按下 Button 後要執行的內容要放在大括號內，程式碼如下。

```
3   struct ButtonView: View {
4       var imageName = "image01"
5       var body: some View {
6           VStack{
7               Button("Press"){
8                   imageName = "image02"   ⊗ 無法指定到屬性：'self' is immuta...
9               }
10              Image(imageName)
11                  .resizable()
12                  .aspectRatio(contentMode: .fit)
13                  .frame(width: 200)
14          }
15      }
16  }
```

我們使用了 imageName = “image02” 來更換顯示的內容，但得到了一個 Error 。
在宣告 imageName 的前面，加上一個 @State，注意 S 需要大寫，結果如下。

```
3   struct ButtonView: View {
4       @State var imageName = "image01"
5       var body: some View {
6           VStack{
7               Button("Press"){
8                   imageName = "image02"
9               }
10              Image(imageName)
11                  .resizable()
12                  .aspectRatio(contentMode: .fit)
13                  .frame(width: 200)
14          }
15      }
16  }
```

回到畫面測試，此時按下按鈕圖片就會變成 image02，但這個按鈕只有按下第一次會改變圖片，按下第二次後就都沒有反應了。

Thinking Time ： @State 的用途是？

章節筆記

語法疑問：

教學重點：

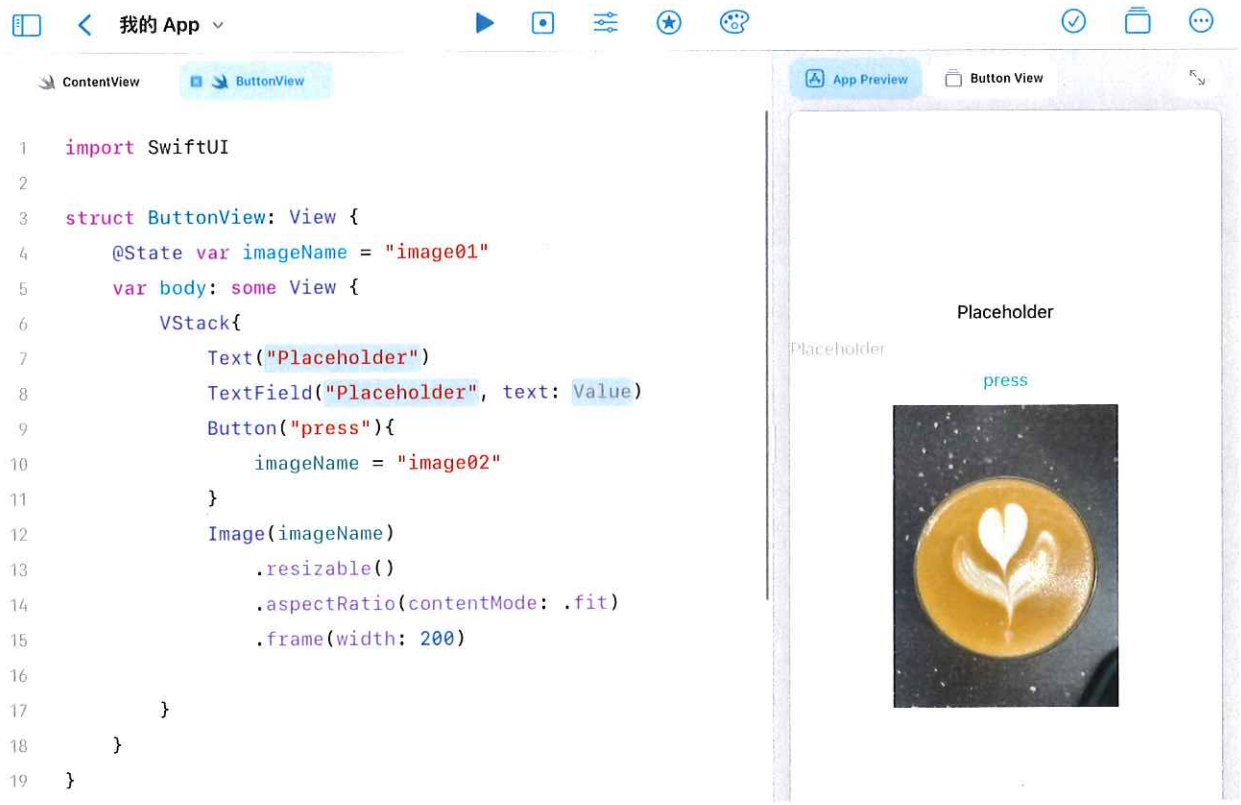
反思回饋：

文字互動 – TextField、Button、Text

新增文字欄位

接下來的任務是，在畫面上新增一個文字與文字欄位，在文字欄位輸入文字並按下按鈕後，文字內容要被更新。

使用元件庫，在按鈕的前方新增一行文字，與文字欄位。



TextField 是一個可以輸入的文字欄位，這個兩個淡藍色底是需要修改內容的位置。第一個位置代表這個輸入欄位的輸入提示，會有淡淡的藍灰色，在 TextField 裡面，如現在**預覽**是 Placeholder，第二位置要填入的是一個變**數**，User 輸入的內容會存到這個變**數**裡。

新增一個變數 input 來儲存 TextField 輸入的內容，並新增一個變數 message 來儲存Text 要顯示的內容。

```
3   struct ButtonView: View {
4       @State var imageName = "image01"
5       @State var input = ""
6       @State var message = "Hello"
7       var body: some View {
8           VStack{
9               Text(message)
10              TextField("Your Name", text: $input)
```

此時 TextField 輸入的內容要和 input 綁定在一起，所以需要在 TextField 的第二個引數 (Argument) input 前面要加上 $ (Dollar Sign)

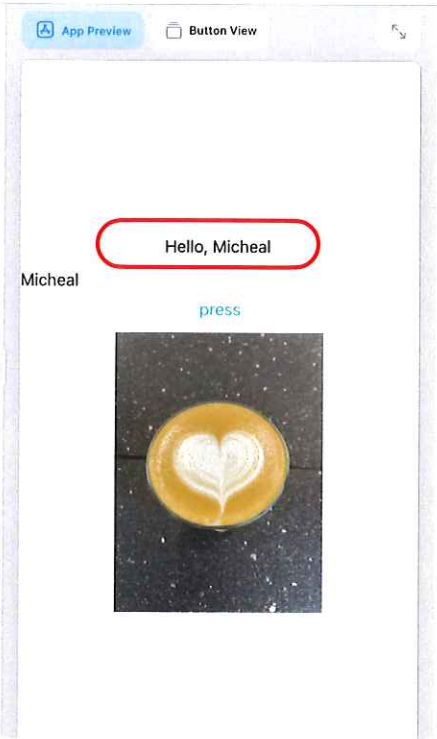
接著在 Button 裡面加入更改 Text 的事件，

```
message = "Hello, " + input
```

試試看，**預覽**，輸入文字，按下 Button。
頁面上方的文字會從 Hello 變成 Hello, Micheal 。

ContentView　ButtonView

```
1   import SwiftUI
2
3   struct ButtonView: View {
4       @State var imageName = "image01"
5       @State var input = ""
6       @State var message = "Hello"
7       var body: some View {
8           VStack{
9               Text(message)
10              TextField("Your Name", text: $input)
11              Button("press"){
12                  imageName = "image02"
13                  message = "Hello, " + input
14              }..
15              Image(imageName)
16                  .resizable()
17                  .aspectRatio(contentMode: .fit)
18                  .frame(width: 200)
19
20          }
21      }
```



章節筆記

語法疑問：

教學重點：

反思回饋：

條件碼

在按鈕與圖片的畫面互動範例中，Button 按下去之後，圖片只會被改變一次，後來再點選 Button 圖片都不會改變。

若要讓 Button 按下後，顯示的圖片在 image01 與 image02 的圖片之間切換，這時就需要使用到條件碼來判斷在不同條件下，imageName 要給不同的內容。

在 Swift 中條件碼的語法如下：

如果 『條件1』 為 true，會執行 『執行內容1』 ，如果為false 會執行『執行內容2』

```
if 條件1 {
    執行內容1
} else {
    執行內容2
}
```

在Button中加入條件碼：

加入條件碼的語法並點選 if 後，整個條件碼的語法會變成灰底或是藍色底，此時大括弧可以下拉，讓下方的程式碼包含在這個條件碼之內，再次點選 if，上方會出現可以插入else語句的快捷鍵。

```
Button("press"){
    if imageName == "image01" {
        |
    }..
    imageName = "image02"
    message = "Hello, " + input

}
```

完成 Button 內的條件碼，測試看看圖片是否能順利在 image01 與 image02 之間切換。

```
Button("press"){
    if imageName == "image01" {
        imageName = "image02"
    } else {
        imageName = "image01"
    }
    message = "Hello, " + input
}..
```

終極密碼 APP

猜 1 ~ 100 的猜數字遊戲，玩家透過猜測一個電腦出的隨機數字，根據提示逐步縮小答案範圍，直到猜中為止。

小組討論

Q1：終極密碼 APP 畫面規劃 ？需要用到的畫面元件有哪些 ？

Q2：哪些畫面元件需要加入邏輯的判斷？

Q3：哪些畫面元件需要跟其他的畫面元件互動？

目前介紹過的畫面元件

1. Text( ) 顯示文字：文字可以是固定輸入好的，也可以是一個變數。
2. Button( ) 按鈕：可以按下按鈕執行某些事件，也可以在其中加入條件碼來依照不同情形，執行不同的內容。
3. TextField( ) 文字欄位：讓使用者能輸入文字或數字，並可以取用輸入的內容。

終極密碼 APP 畫面規劃

遊戲畫面規劃

先將遊戲畫面元件加入，並宣告變數來存取玩家猜的數字與猜測結果。

1. 新增一個 App。
2. 加入畫面元件 Text( ) 內容為遊戲標題與遊戲規則。
3. 在 body 外宣告變數 guessNumber ，型別為整數 Int ，初始值設定 0。
4. 使用畫面元件庫加入TextField( ) ，讓玩家輸入猜測的數字。
   原本 TextField( ) 的第二個引數 text: 能加入的變數型別是 String ，須修改為 value: ，才能存取型別為整數 Int 的變數，guessNumber 前記得加上 $ 將其與 TextField 綁定在一起，當 TextField 內容有變化時 guessNumber 會自動更新。最後為了確保玩家輸入的內容為整數，還需要指定輸入內容的格式 format: .number 。

```
TextField("Placeholder", text: Value)

TextField("guess a number", value: $guessNumber, format: .number)
    .keyboardType(.numberPad)
```

5. 在 TextField( ) 可以加入 .keyboardType(.numberPad) 的屬性，讓玩家從數字鍵盤輸入數字，提高輸入的準確性。
6. 加入畫面元件 Button () ，玩家輸入數字後，要判斷輸入結果。
7. 宣告一個變數 guessResult 來儲存猜數字的結果，並加入畫面元件 Text() 來猜數字的結果。

ContentView

```
1  import SwiftUI
2
3  struct ContentView: View {
4      @State var guessNumber: Int = 0
5      @State var guessResult: String = ""
6      var body: some View {
7          VStack {
8              Text("Guess the Number!")
9              Text("猜一個1~100之間的數字")
10
11             TextField("guess a number", value:
                $guessNumber, format: .number)
12                 .keyboardType(.numberPad)
13
14             Button("Guess") {
15                 Action
16             }
17
18             Text(guessResult)
19         }
20     }
21 }
```

App Preview

遊戲邏輯判斷 -- 按鈕與條件碼

開始遊戲前，要一個遊戲中要猜的目標數字，接著要在 Button 中加入遊戲判斷的程式碼，並修改遊戲的顯示結果。

1. 宣告一個變數 ans 並由電腦隨機生成一個型別為整數的數字 Int.random，範圍是1 ~ 100 (in: 1...100)，之後要在遊戲中加入一個重新開始遊戲的按鈕，ans 的內容會被更新，所以需要在 var 前面加@State。

   ```
   @State var ans = Int.random(in: 1...100)
   ```

2. 在 Button 中加入判斷遊戲結果的程式碼，判斷使用 if / else if / else 的語句來寫。如果猜的數字等於答案，guessResult 設定為“🎉恭喜猜對了🎉”；或者如果猜的數字大於答案，guessResult 設定為”猜的數字太大了！"；否則 guessResult 設定為"猜的數字太大了！"。

   ```
   Button("Guess") {
       if guessNumber == ans {
           guessResult = "🎉恭喜猜對了🎉"
       } else if guessNumber > ans {
           guessResult = "猜的數字太大了！"
       } else {
           guessResult = "猜的數字太小了！"
       }
   }..
   ```

3. 此時遊戲已經可以正常運作了。結束遊戲後，需要一個 Button 讓遊戲重新開始。加入一個 Reset 的 Button ( )，重新給一個要猜的數字，並把顯示結果與儲存玩家輸入答案的值改回預設值。

   ```
   Button("Reset") {
       ans = Int.random(in: 1...100)
       guessResult = ""
       guessNumber = 0
   }..
   ```

4. 試玩看看遊戲。

Thinking Time ：這個遊戲哪裡可以設計得更好？

更完整的遊戲設計

Challenge：加入目標數字提示範圍

1. 宣告變數 max 與 min 來儲存目前的範圍。

2. 加入一個畫面元件 Text ( ) ，內容是標示現在的範圍。在 Text 裏面只能放型別為String 的內容，若要把非變數放在 String 裡面，要用 \() 把變數放在裡面，變數的內容會被轉成 String ，在下方例子中 \(max) 把 Int 轉成 String。

```
Text(" \(min) < 目標數字 < \(max)")
```

3. 在 Button("Guess") 中修改條件碼的內容，重新設定 max 與 min 的值。

4. 測試與修改

Challenge：計算遊戲次數

1. 宣告一個變數計算猜測次數。

2. 更新猜測次數。

3. 顯示猜測次數。

4. 測試與修改

加強背景知識

學習程式設計 1 — 條件碼
學習程式設計 2 — 變數

章節筆記

語法疑問：

教學重點：

反思回饋：

猜燈謎 APP

陣列資料 Array

在一個猜燈謎的app中，題目會有很多題，為了方便儲存與取用資料，將猜燈謎的題目與答案使用 Array 儲存。

```
let question:[String] = ["尖尖嘴,細 細腿,狡猾多疑拖大尾", "不走光跳,吵吵鬧鬧,吃蟲吃糧,功大過小", "為我打你,為你打我,打破你的皮,流出我的血", "唱歌不用嘴,聲音真清脆,嘴尖像根錐,專吸樹汁水"]
let answer:[String] = ["狐狸", "麻雀", "蚊子", "蟬"]
```

上方儲存猜謎資料的 question 與 answer 中，儲存的類型是 Array 用 [] 表示，但其內容物一定要給類型，在這裡 Array 由 String 組合而成，所以 [] 要放 String 成為 [String] 。

要取用 Array 的資料要用 [] 裡面包含 index，Array 的第一個資料 index 為 0。

例如：question[0]

在 Swift 中，使用 let 來宣告的是常數，它的內容無法在宣告後被更改。使用 var 關鍵字來宣告的是變數，它的值可以在程式中更改。

加入畫面元件

猜燈謎的遊戲中，畫面需要：

1. 一個畫面元件 Text( ) 顯示題號，另一個畫面元件 Text( ) 取用 question 陣列資料顯示題目，並宣告一個變數紀錄題號與取用陣列資料。
   在 View 的 body 外面宣告變數 index，在 View 的 body 裡面加入兩個 Text( ) 顯示題號與顯示題目：

```
3   struct ContentView: View {
4       let question:[String] = ["尖尖嘴,細 細腿,狡猾多疑拖大尾",
         "不走光跳,吵吵鬧鬧,吃蟲吃糧,功大過小", "為我打你,為你打我,打破你的皮,流出我的血",
         "唱歌不用嘴,聲音真清脆,嘴尖像根錐,專吸樹汁水"]
5       let answer:[String] = ["狐狸", "麻雀", "蚊子", "蟬"]
6       @State var index = 0
7       var body: some View {
8           VStack {
9               Text("第\(index+1)題")
10              Text(question[index])
11
12          }
13      }
14  }
```

因為陣列的第一個內容 index 為 0，所以將 index 的初始值設定為 0，將 index+1 就是題號。使用 \() 字串插值將 index+1 的型別由 Index 轉換成 String

2. 畫面元件 TextField( )，讓玩家輸入猜測的答案，宣告一個型別為 String 變數來儲存輸入的內容。
3. 畫面元件 Button( )，按下按鈕後判斷答案正確性，與是否需要進行到下一題。
4. 畫面元件 Text( ) 顯示結果，宣告一個型別為 String 變數儲存結果。

```
1   import SwiftUI
2
3   struct ContentView: View {
4       let question:[String] = ["尖尖嘴,細 細腿,狡猾多疑拖大尾",
         "不走光跳,吵吵鬧鬧,吃蟲吃糧,功大過小", "為我打你,為你打我,打破你的皮,流出我的血",
         "唱歌不用嘴,聲音真清脆,嘴尖像根錐,專吸樹汁水"]:
5       let answer:[String] = ["狐狸", "麻雀", "蚊子", "蟬"]
6       @State var index:Int = 0
7       @State var input:String = ""
8       @State var result:String = ""
9
10      var body: some View {
11          VStack {
12              Text("第\(index+1)題")
13              Text(question[index])
14              TextField("猜一種生物", text: $input)
15              Button("猜對了嗎？") {
16                  Action
17              }
18              Text(result)
19          }
20      }
21  }
```

Thinking Time ：Button 裡面需要什麼樣子的邏輯判斷？

遊戲邏輯判斷

在 Button 內加判斷式來判斷輸入答案是否正確，若輸入答案正確，結果顯示答對了，並要進行到下一題 index 要加 1，否則結果顯示再猜猜看。

```
Button("猜對了嗎？") {
    if input == answer[index]{
        result = "🎉答對了🎉
        index += 1
    } else {
        result = "再猜看看""
    }
}
```

測試一下遊戲，看看遊戲是否能順利進行到最後？

Thinking Time ：當遊戲進行到最後一題時，遊戲 Crashed 了！畫面中顯示索引超出範圍？發生什麼事？可以怎麼修正？

```
2
3   struct ContentView: View {
4       let question:[String] = ["尖尖嘴,細 細腿,狡猾多疑拖大尾",
         "不走光跳,吵吵鬧鬧,吃蟲吃糧,功大過小", "為我打你,為你打我,打破你的皮,流出我的血",
         "唱歌不用嘴,聲音真清脆,嘴尖像根錐,專吸樹汁水"]
5       let answer:[String] = ["狐狸", "麻雀", "蚊子", "蟬"]
6       @State var index:Int = 0
7       @State var input:String = ""
8       @State var result:String = ""
9
10      var body: some View {
11          VStack {
12              Text("第\(index+1)題")
13              Text(question[index])
14              TextField("猜一種生物", text: $input)
15              Button("猜對了嗎？") {
16                  if input == answer[index]{
17                      result = "🎉答對了🎉"
18                      index += 1
19                  } else {
20                      result = "再猜看看"
21                  }
22              }
23              Text(result)
24          }
```

索引超出範圍

定義 struct 結構

在前面**我**們將猜燈謎的題目與答案，儲存在兩個 array 中，每一次遊戲進行的時候都只能從**我**們存好的問題中，第一題進行到最後一題，沒辦法隨機出題，所以**我**們需要使用 Struct 結構來儲存資料。

定義一個名為 Riddle 的結構，注意宣告 struct 的名稱時，需要使用駝峰式大寫命名，宣告變**數**與 function 時，使用駝峰是小寫命名。注意宣告 Riddle 的 struct 的位置要在 View 的 struct 之外。

在 struct 中宣告了兩個型別為 String 的常**數**，在宣告 struct 時，內容可以不用給定初始值，在初始化的時候再給初始值就好：

```
1   import SwiftUI
2
3   struct Riddle {
4       let question: String
5       let answer: String
6   }
```

使用struct 中宣告的 Riddle 是一種類型，將猜燈謎的題目與答案使用 Riddle 的類型來儲存，使用方法如下：

```
let q1 = Riddle(question: "尖尖嘴,細細腿,狡猾多疑拖大尾", answer: "狐狸")
let q2 = Riddle(question: "不走光跳,吵吵鬧鬧,吃蟲吃糧,功大過小", answer: "麻雀")
let q3 = Riddle(question: "為我打你,為你打我,打破你的皮,流出我的血", answer: "蚊子")
let q4 = Riddle(question: "唱歌不用嘴,聲音真清脆,嘴尖像根錐,專吸樹汁水", answer: "蟬")
```

在遊戲畫面的 Struct 中，將題目宣告為一個 Array 的變**數**，Array 內容的型別為**我**們宣告的 Riddle，內容放q1、q2、q3、q4，

```
@State var riddles:[Riddle] = [q1, q2, q3, q4]
```

取用 riddles 中第一個謎題的問題：

```
riddles[0].question
```

取用 riddles 中第一個謎題的答案：

```
riddles[0].answer
```

將前面的程式碼顯示問題的地方修改成 Text(riddles[index].question) ，判斷答案 answer[index] 改為 riddles[index].answer。

為了避免最後一題猜中答案之後，index 會超出範圍，在 index += 1 包含在一個判斷式中。對 array 使用 .count，可以得知 array 裡有幾個內容。

以前面宣告的 riddles 來舉例 riddles.count 會是 4，riddles 最後一個內容的 index 是3，所以判斷 index 要不要 +1 決定於 index < (riddles.count –1)。

```
Button("猜對了嗎？") {
    if input ==  riddles[index].answer  {
        result = "🎉答對了🎉"
        if index < (riddles.count - 1 ){
            index += 1
        }
    } else {
        result = "再猜看看"
    }..
}
```

此外還可以將題號顯示修改成 Text("第 \(index+1)/\(riddles.count) 題")

測試看看遊戲，看是否能順利進行！

加強背景知識

學習程式設計 1 — 條件碼
學習程式設計 2 — 變數、陣列

進階__定義 Struct 結構：學習程式設計 2 — 初始化

章節筆記

語法疑問：

教學重點：

反思回饋：

如何分享 App 給其他台 iPad ？

其他問題？

作者：Michael Pan
版權：Zencher Co. Ltd. 2023, 並分享給所有與會老師使用