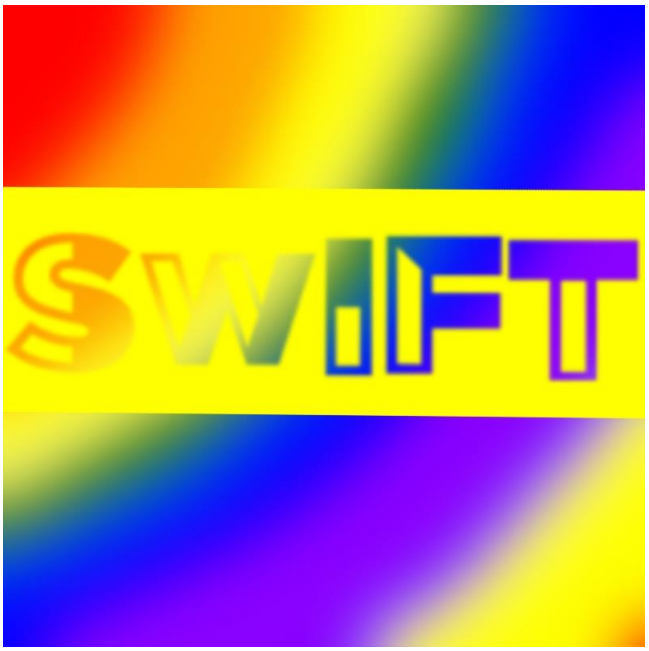
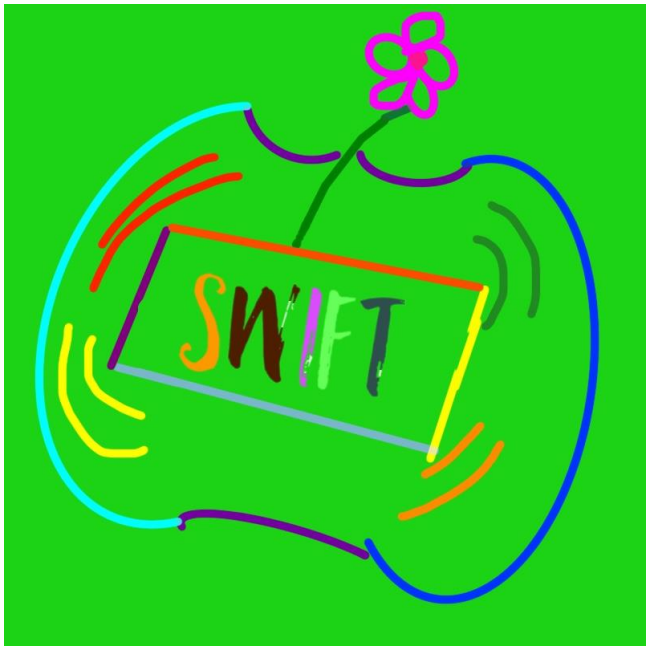
嘉義市第 37 屆中小學科學展覽會
作品說明書





科　　別：生活與應用科學科(1)(機電與資訊)

組　　別：國小

作品名稱：Swift ⇔ app

關 鍵 詞：Swift、app

編　　號：

摘要

使用 Swift 程式語言、設計 app 循環四階段法，搭配 Xcode 寫出 iOS 的 app。此 app 利用自創的「三色磁鐵位置筆記法」介紹 Swift 程式語言 4.2 版，讓初學者能經由點選就能簡單的學習程式語言的基礎，經由 macbook pro 裡的 Xcode 的模擬器功能證實此 app 真的能 work。

壹、研究動機

在學校上課時，老師說 12 年國教上路後，大家都要學程式語言，雖然上課有教 scratch，但是我們幾個好朋友還想要一起學其他的程式語言。後來在新聞看到日本小學生程式競賽，看到他們這麼厲害的寫出自己的 app，讓我們覺得要不落人後，利用學習的 Swift 程式語言寫一個 app 來推廣學習 Swift 程式語言。

貳、研究目的

一、視覺化 Swift 程式語言

二、規劃 app 的使用者介面

三、製作 app 的原型

四、評估 app

五、設計 app icon

參、研究設備及器材

一、桌上型電腦

二、Xcode（開發蘋果 OS 的應用程式）

三、ipad pro

四、procreate（繪圖 app）

五、Google 翻譯

六、Send Anywhere（跨平台文檔傳輸 app）

七、小畫家

八、apple pencil

九、macbook pro

十、Pages(文書軟體)

肆、研究過程或方法

想要做一個 app 教別人程式語言之前，自己得要先腳踏實地的學習過才行。我們需要先對英文輸入法有一定程度的練習，再來挑戰後面的研究。

一、手腦並用：我們問老師、使用 Google 翻譯，以瞭解 Swift 4.2 版的相關定義，並動手打字將程式碼輸入到 Xcode 的 playground，來觀察實際輸入程式碼所呈現的基礎語法結果以及過程中所遇到的問題。接下來進入設計 app 的循環階段，一步步的設計 app。

二、在研究過程一中，我們將 Swift 程式視覺化後，接下來就是把相關的資料圖檔做編輯，再傳到 Xcode 做後續處理，從 Xcode 挑選適當的元件來規劃使用者介面。

三、在 Xcode 裡，實際製作 app 的原型。

四、在 Xcode 裡，以模擬器評估 app 的效果。

五、使用 procreate 製作 app icon

伍、研究結果與討論

實驗一：視覺化 Swift 程式語言

詢問老師、使用 Google 翻譯，而瞭解 Swift 程式語言的定義，並將程式碼輸入到 Xcode 的 playground，觀察實際輸入程式碼所呈現的基礎語法結果以及過程中所遇到的問題之後，我們發現內容很多。要做出一個 app 來介紹 Swift 的話，一定要做適當取捨。接下來進入設計 app 的循環階段，要解決這些問題。

app 的循環階段一：腦力激盪

目的：讓大家學習 Swift 程式語言變得更簡單有趣、視覺化、生活化。

想法：大家腦力激盪，看要用哪些方法能讓學習 Swift 是視覺化、生活化的。

適用對象：因為我們的目的，很明顯要針對初學程式語言或是對 Swift 不熟的人。多用一些視覺化的技巧、文字做連結，應該可以幫助學習，提高學習效果才對。這也表示不能介紹的太深入。

聚焦目標：前面討論很多而且做了很多功課之後，有很多想法跑出來，覺得很難做出取捨，所以我們覺得要訂出標準來決定哪些想法要留下來，寫出 app 的說明來清楚的定義它的目的，可以讓我們判斷哪些是好的想法，選出最有趣的出來。

app 說明：這個 app 可以：輕鬆學習、認識 Swift 程式語言

因為：簡單、視覺化

重新思索：最後再想一想，照著之前的藍圖來做的話，這個 app 有沒有比其他 app 更有創意或是更棒的優點？大家會喜歡它嗎？

經過腦力激盪後，我們覺得要視覺化 Swift 的龐大內容，將之筆記化的摘要重點是不錯的方向。在 macOS 的 Pages 視覺化 Swift 程式語言的成果請參照圖 5-1～圖 5-10。

一
Constant（常數）:它的值不能被改變。就像圓周率的值是不能改變的。

宣告的關鍵字或結構: let

程式碼範例:
```
let 圓周率 = 3.14159
             常數為Double
let threeWatches: String = "⌚⌚⌚"
                  值的型態
```
二
Variable（變數）:它的值能被改變。就像籃球比賽的分數是可以改變的。

宣告的關鍵字或結構: var

程式碼範例:
```
var 😀 = 9
😀 = 😀 + 11
var 體重, 身高: Double; var 姓名 = ""
  變數體重、身高都為Double          空字串
```

三
String（字串）:文本資料。

宣告的關鍵字或結構: ""

程式碼範例:
```
var 😎 = 555, 😜 = 666
print("太陽眼鏡與鬼臉相加等於\(😎+😜)")
                          字串插值 → 1221
```
四
Tuple（元組）:有順序(值都有索引)的將多個值(可以是不同型態)組成一個複合值。

宣告的關鍵字或結構: ()

程式碼範例:
```
let manyValues = (順位:1, 2.3, "OK", 45)
                索引   0,   1,   2  , 3
print("\(manyValues.3)")
                    索引3 → 45
print("\(manyValues.順位)")
                     項目名稱 → 1
```

圖 5-1、整理在 Pages 的筆記截圖之一

```
五
Array（陣列）:有順序(值都有索引)的儲存同一種型態的集合資料

宣告的關鍵字或結構: Array<>  or  []

程式碼範例:
var a陣列: Array<String>
var b陣列: [Int]

var someInts = [Int]()
                空的整數陣列
someInts.append(3)
           添加1個項目

var threeDoubles = Array(repeating: 0.0, count: 3)
                                重複的值    計數 → [0.0, 0.0, 0.0]
threeDoubles = []
               指派為空陣列

var shoppingList = [“eggs”, “milk”, “魚肉”, “豬肉”]
                索引   0        1      2      3
print(“the shopping list contains \(shoppingList.count) items”)
                                                    計數 → 4
shoppingList.append(“vegetable”)
               添加1個項目
shoppingList += [“西瓜” ,”牛肉”, “地瓜”]
              添加1個以上的項目
var 陣列的第一個項目 = shoppingList[0]
                                “eggs”
shoppingList[0] = “two pens”
           第1個項目變更為其他項目
shoppingList[4…6] = [“oranges”, “apples”]
            第5~7個項目變更為其他項目
shoppingList.insert(“gold ring”, at: 2)
              在索引2插入1個項目
let 金戒指 = shoppingList.remove(at: 0)
               指派陣列的索引1的項目為常數金戒指的值，並移除陣列的第1個項目
let 蘋果 = shoppingList.removeLast()
              指派陣列的最後1個項目為常數蘋果的值，並移除陣列的最後1個項目
```

圖 5-2、整理在 Pages 的筆記截圖之二

六
Set（集合）:沒有順序且不重複的儲存同一種型態的集合資料。

宣告的關鍵字或結構: Set<> or []

程式碼範例(Array（陣列）的dot syntax，Set也可以用。):
```
var a集合: Set<Int> = [1, 2, 3, 4, 5, 6]
a集合 = []
        指派為空集合
a集合 = [1, 2, 3, 4, 5, 6, 7]
a集合.removeAll()
     集合裡的值都移除
let oddDigits: Set = [1,3,5,7,9]
let evenDigits: Set = [0,2,4,6,8]
let singleDigitPrimeNumbers: Set = [2,3,5,7]

oddDigits.intersection(evenDigits).sorted()
                    交集為       []
oddDigits.symmetricDifference(singleDigitPrimeNumbers).sorted()
                    對稱差的集合為 → [1,2,9]
oddDigits.union(evenDigits).sorted()
                    聯集為 → [0,1,2,3,4,5,6,7,8,9]
oddDigits.subtracting(singleDigitPrimeNumbers).sorted()
                    差集為 →  [1,9]
```

七
Dictionary（字典）:沒有順序的儲存鍵值配對key:value(key要同一種型態，value要同一種型態)的集合資料。

宣告的關鍵字或結構: Dictionary<Key, Value> or [Key: Value]

程式碼範例:

```
var namesOfIntegers = [Int: String]()
                       key為Int, value為String的空字典變數
namesOfIntegers[16] = “fifteen”
             新增key-value配對
namesOfIntegers[16] = “sixteen”
             因為已有該配對，所以是變更key-value配對的value
namesOfIntegers[11] = “eleven”
namesOfIntegers[11] = nil
                      value指派為nil,即可移除該key-value配對
namesOfIntegers = [:]
             指派為空字典
```

圖 5-3、整理在 Pages 的筆記截圖之三

八
For-in loops: 迭代一個排序的值。也就是重複一個範圍的值。是控制流程的一種。

宣告的關鍵字或結構: for 排序的項目名稱 in 排序 {}

程式碼範例:

```
let names = [“Bill”, “Bach”, “Jane”]
for name in names {
    排列的項目名稱，不需要宣告
  print(“hello, \(name)!”)
}
         hello, Bill!
         hello, Bach!
         hello, Jane!

let numberOfLegs=[“spider”: 8, “ant”: 6, “dog”: 4]
for (animalName, legCount) in numberOfLegs {
     字典的項目名稱以tuple的方式呈現，不需要宣告
 print(“\(animalName)s have \(legCount) legs”)
}
                  ants have 6 legs
                  dogs have 4 legs
               spiders have 8 legs

for index in 1…5 {
 print(“\(index) times 5 is \(index * 5)”)
}              1 times 5 is 5
               2 times 5 is 10
               3 times 5 is 15
               4 times 5 is 20
               5 times 5 is 25

let base=3
let power=5
var answer =1
for _ in 1…power {
  省略排列的值而不用。則1...power表示計數
  answer *=base
}
print(“\(base) to the power of \(power) is \(answer)”)
             3 to the power of       10 is 243
```

圖 5-4、整理在 Pages 的筆記截圖之四

九
While: 先檢查條件，符合條件才會執行程式。是控制流程的一種。

Repeat-while：先執行程式，才檢查條件是否符合，符合時就繼續下一個循環。

宣告的關鍵字或結構:

```
while 條件 {執行程式}
repeat {執行程式} while 條件
```

程式碼範例:

```
var x = 25
while x < 24 {
 x = x * 2
}
print(x)
      25

var x = 25
repeat {
 x = x * 2
} while x < 24
print(x)
      50
```

十
If: 先檢查條件，符合條件才會執行程式。是控制流程的一種。
else: 都不符合條件才會執行程式。if句不一定要有else
else if: 第二個以上的檢查條件，符合條件才會執行程式

宣告的關鍵字或結構:

```
if 條件 {執行程式}
if 條件 {執行程式} else {執行程式}
if 條件 {執行程式} else if 條件 {執行程式}
if 條件 {執行程式} else if 條件 {執行程式} else {執行程式}
```

程式碼範例:

```
var temperatureInFahrenheit = 90

if temperatureInFahrenheit <= 32 {
 print(“cold. wear a scarf”)
} else if temperatureInFahrenheit >= 86 {
 print(“warm. wear sunscreen”)
} else {
print(“not cold. wear a t-shirt”)
}
        warm. wear sunscreen
```

圖 5-5、整理在 Pages 的筆記截圖之五

十一

Switch: 將值與多個情況比對，有符合的情況就執行該情況的程式；若是情況都不符合就執行default的程式。是控制流程的一種。

宣告的關鍵字或結構:

```
switch 值 {
case 情況1: 執行的程式1
case 情況2, 情況3:執行的程式2或程式3
default: 執行程式
}
```

程式碼範例:

```
let someCharacter: Character = “z”
switch someCharacter {
case “a”, “A”:
  print(“the first letter of the alphabet”)
case “z”:
  print(“the last letter of the alphabet”)
default:
  print(“ some other character”)
}
```

```
the last letter of the alphabet
```

```
let 便當價錢=60
switch 便當價錢 {
case 0:
  print(“免費”)
case 1..<10:
  print(“超便宜便當”)
case 10..<50:
  print(“價錢便宜的便當”)
default:
  print(“不便宜的便當”)
}
```

```
不便宜的便當
```

圖 5-6、整理在 Pages 的筆記截圖之六

```
let somePoint = (1,2)
switch somePoint {
case (0,0):
  print("\(somePoint)在原點")
case (_,0):
  print("\(somePoint)在x軸上")
case (0,_):
  print("\(somePoint)在y軸上")
default:
  print("\(somePoint)不在x軸或y軸上")
}
                (1,2)不在x軸或y軸上


let anotherPoint = (1,0)
switch anotherPoint {
case (let x,0) where x == 0:
                額外的條件
  print("\(anotherPoint)在原點")
case (let x,0):
  值綁定到暫時的常數
  print("在x軸上，其值是 \(x)")
case (0, let y):
  print("在y軸上，其值是 \(y)")
case let (x, y):
  值綁定到暫時的常數
  print("不在x軸或y軸上的點 (\(x), \(y))")
}
         在x軸上，其值是 1
```

圖 5-7、整理在 Pages 的筆記截圖之七

十二
Continue:不做該循環在continue的後續部分，而開始下一個循環。是控制流程的一種。

宣告的關鍵字或結構: continue

程式碼範例:

```
for 不印出三的倍數 in 1...7 {
 if 不印出三的倍數 % 3 == 0 {
  continue
 }
 print(不印出三的倍數)
}
```

```
1
2
4
5
7
```

十三
Break:立刻的結束執行控制流程。是控制流程的一種。

宣告的關鍵字或結構: break

程式碼範例:

```
for 不印出大於二的數 in 1...7 {
 if 不印出大於二的數 > 2 {
  break
 }
 print(不印出大於二的數)
}
```

```
1
2
```

圖 5-8、整理在 Pages 的筆記截圖之八

十四
Functions(函數): 執行特定任務的獨立的多塊程式碼。

宣告的關鍵字或結構:

```
func 函數名稱(參數標籤 參數名稱: 參數型態) -> 返回值型態 {
        呼叫函數時用    在函數定義裡使用。呼叫函數時不用
 執行的程式
 return 返回值

}

func 函數名稱(_ 參數名稱: 參數型態) -> 返回值型態 {
        呼叫函數時，不必寫參數標籤
 執行的程式
 return 返回值

}

func 函數名稱(參數名稱: 參數型態) -> 返回值型態 {
       是參數標籤也是參數名稱
 執行的程式
 return 返回值

}

func 函數名稱(參數名稱: 參數型態) -> 返回值型態 {
              可以沒參數          可以沒返回值
 執行的程式
 return 返回值

}

func 函數名稱(參數名稱: 參數型態...) -> 返回值型態 {
  可變參數:接受特定同一型態的0個以上的值的參數。一個函數只能有一個可變參數。
 執行的程式
 return 返回值

}
func 函數名稱(參數名稱: inout 參數型態) -> 返回值型態 {
    in-out參數:函數能修改參數的值，呼叫函數後的修改仍然存在。
 執行的程式
 執行的程式
 return 返回值

}
```

圖 5-9、整理在 Pages 的筆記截圖之九

程式碼範例:

```
func 沒有參數和返回值() {
 print("123")
}
沒有參數和返回值()
        123


func arithmeticMean(_ numbers: Double…) -> Double {
 var total: Double = 0
 for number in numbers {
 total += number
 }
 return total/Double(numbers.count)
}
print(arithmeticMean(1,2,3,4,5))
                          3.0
print(arithmeticMean(3,8.35,18.65))
                         10.0


func 兩數交換(_ a: inout Int, _ b: inout Int) {
 let temporaryA = a
 a = b
 b = temporaryA
}
var 某數 = 3
var 另一數 = 107
兩數交換(&某數, &另一數)
       &直接加在in-out參數的變數前
print("某數現在是 \(某數), 另一數現在是 \(另一數)")
                  107                    3
```

十五

Optional: 允許常數或是變數可以沒有值，沒有值就是代表nil。

宣告的關鍵字或結構: ?

程式碼範例:

```
var red: Int?
       預設值為nil
```

圖 5-10、整理在 Pages 的筆記截圖之十

【實驗結果】:

（一） 我們找了 15 個重要的程式語言的名詞，做了簡單的摘要定義、解釋，其解釋主要是以字體紅色（綠色及黑色字體為輔）來化繁為簡而且層次分明、由上而下，雖然宛如磁鐵一般緊貼，但是讓人一目瞭然，不會把解釋與正文搞混；程式碼的輸入結果也用紅色字體且對齊的放在該程式碼區塊的最後面，看起來非常整齊。或是用紅色的字體「突顯」要特別注意的地方。與一般教授程式語言的書籍比起來，它們很喜歡寫長長的程式碼，解釋的位置與程式碼有好一段距離，要花時間交互比對，我們的方法比較省時、直覺。而 Swift 原有的解釋程式碼的方式「//」或是「/＊　＊/」，讓我們覺得少了一些解釋的彈性。經過不斷改良採用的筆記上色法，我們稱為「三色磁鐵位置筆記法」，這很符合平常上課塗色畫重點的方法。如圖 5-1～圖 5-10。

（二） 在相關的程式碼的命名，發現可以用繁體中文、表情符號、符號、圖示等多元的字元命名之後，我們就嘗試把這些命名方式代入改寫程式碼，讓學習者發現原來命名可以這麼有趣。除了因為英文程度有待加強之外，使用繁體中文的字元做出符合程式碼目標的命名還可以讓人輕易的知道這個程式碼是在做什麼，例如圖 5-6 最後一段的「便當價錢」的程式碼。如果是其他國家的學習者，使用該國的語言當做命名的字元也是可以的。

（三）程式碼的選用也是很重要的，因為對於初學者來說看到太多程式碼在 app 上，應該是很頭痛的，所以我們將程式碼簡化、合併，用最少的程式碼直覺的教導 Swift 程式語言的觀念，並做出取捨來簡化教導的內容讓初學者容易學習，改寫過的程式碼當然都有在 Xcode 的 playground 測試可以使用。

（四）為了有朝一日能讓 app 上架，使用 office 的軟體製作可能會有字體侵權問題，所以使用 macOS 的 Pages 文書軟體做 Swift 的視覺化資料再加以編輯。

（五）實驗一裡使用的 app 的循環階段一的腦力激盪，讓我們能聚焦在 app 目的、想法、適用對象，經過重新思索後更能堅持設計 app 的初衷。

實驗二：規劃 app 的使用者介面

將實驗一的視覺化資料做適當編輯後，將其圖檔傳入 Xcode 的資料夾「Assets.xcassets」以做後續使用。再來是進入第二個 app 的循環階段。

app 的循環階段二：規劃

UI 及 UX：我們應該在 Xcode 設計出好的使用者介面，這樣才會有好的使用者體驗。在 Xcode 上設計，我們想要設計出使用者只要簡單的點選、點選，然後發自內心驚嘆聲的 wow 的效果的 app。

設計：我們希望設計出來的 app 夠簡單又有自己的特色，所以太多花俏的顏色會無法讓使用者聚焦目標，設計出來的圖像、文字介紹要能吸引使用者，留下深刻的目標。

【實驗結果】：

（一） 根據 app 的循環階段二的規劃，我們覺得 Xcode 的「Navigation controller」、「library」的「Button」元件及「Image View」元件、segue 應該可以滿足需求。

（二） 使用小畫家來編輯實驗一的 10 個視覺化圖檔資料，將之分割成 21 個圖檔後存入 Xcode 的資料夾「Assets.xcassets」以做後續使用。

實驗三：製作 app 的原型

由實驗二，決定在 Xcode 使用 view controller、Navigation controller、「Button」元件、「Image View」元件再搭配 segue 以及編輯好的視覺化圖檔，製作 app 的原型，進入第三個 app 的循環階段。

app 的循環階段三： 製作 app 的原型

我們使用 macbook pro 在 Xcode 製作 app 原型，在 Xcode 的檔案「Main.storyboard」的 view controller 嵌入「Navigation controller」，然後使用「library」的「Button」元件，做出適當數量的按鈕。再來是新增足夠數量的 view controller，在每個 view controller 新增「Image View」元件來置入實驗一的編輯過的視覺化圖檔。只要將按鈕與各個 view controller 做出 segue，而設計出使用者只要簡單的點選、點選…，就可以輕鬆學習的 app。如圖 5-11。

圖 5-11、在 Xcode 的 storyboard 的 app 原型

【實驗結果】：

（一）如圖 5-11，我們在 Xcode 的檔案「Main.storyboard」的 view controller 嵌入「Navigation controller」，然後在主要的 view controller 使用「library」的「Button」元件，做出 21 個按鈕。再來是另外新增 21 個 view controller，在每個 view controller 新增「Image View」元件來置入在實驗二編輯過的視覺化圖檔，並且將按鈕與對應的 view controller 以 segue 連結，如此設計讓使用者只要簡單的點選按鈕，就可以連結到想學習的部分，學習完再按返回鍵就能回到主要的使用者介面，繼續輕鬆學習。

實驗四：評估 app

app 的循環階段四： 評估 app

再來是測試原型。尋找初學程式語言的同學、朋友來進行測試，向他們展示 app 原型，讓他們使用 Xcode 的模擬器（如圖 5-12）來體驗，觀察他們的體驗後再提出詢問。

在這個過程中，我們要仔細觀察他們是否知道要按哪個按鈕？是否在某個操作中，會有疑惑的表情？他們喜歡這個 app，或是在操作中有露出微笑之類的表情？

他們體驗 app 後，請問他們覺得這個 app 有哪些優缺點？這個 app 能幫助他們輕鬆的學習 Swift 程式語言？會希望此 app 提供哪些功能？收集好使用者的測試資料後，回到第一個腦力激盪階段，以重複 app 的設計階段。

圖 5-12、app 在 Xcode 的模擬器的使用者介面

【實驗結果】:

（一） 由圖 5-12，我們製作出簡單的使用者介面，只要點選想學習的按鈕，就可以連接到已編輯好的圖檔來學習。經由使用者的回饋表示這種方式的確很簡單，如果要讓 app 更完美的話，可以把 app 加以遊戲化，會讓學習 Swift 程式語言更有趣。

實驗五：設計 app icon

App icon 是很重要的門面，我們在 ipad pro 上以第一名的繪圖軟體 procreate 及 apple pencil 來製作，先設定好 1024×1024 像素的畫布，再發揮創意製作 app icon。我們的成果如圖 5-13。

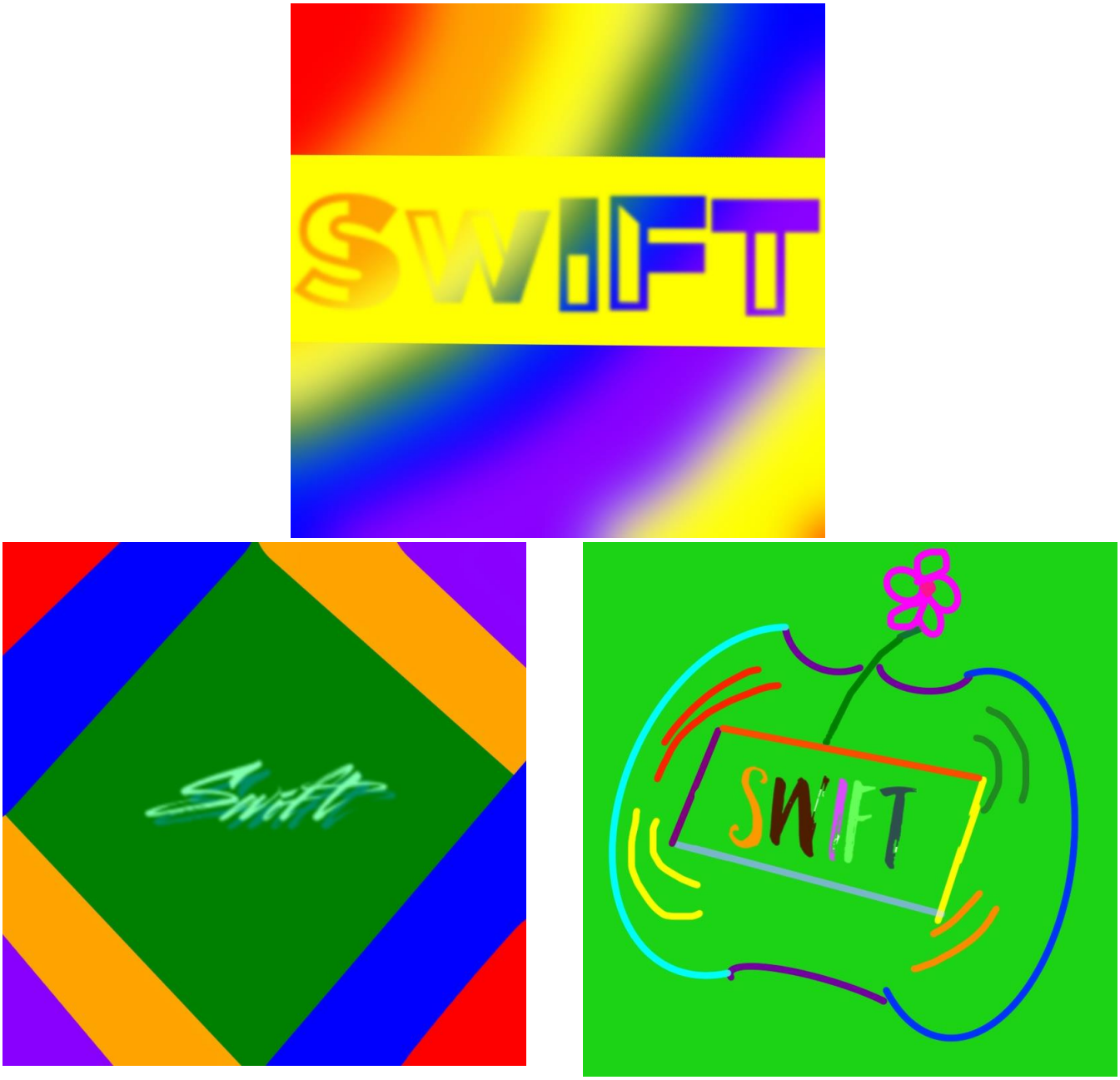


圖 5-13、設計 app icon

【實驗結果】:

（一）由圖 5-13，我們製作出三個 app icon 的圖檔，將之儲存在 Xcode 的資料夾「Assets.xcassets」，等以後 app 上架時就可使用。

陸、結論

（一） 由實驗一，我們利用三個方法試圖讓學習者輕易上手學習。第一個方法是自創「三色磁鐵位置筆記法」做了簡單的摘要定義、解釋，其解釋主要是以字體紅色（綠色及黑色字體為輔）來化繁為簡而且層次分明、由上而下，宛如磁鐵一般緊貼又整齊，讓人一目瞭然，不會把解釋與正文搞混；或是用紅色（主要）、綠色、黑色的字體「突顯」要特別注意的地方。此筆記方法比較省時、直覺，增加許多彈性也符合平常上課塗色畫重點的方法。第二個方法是利用繁體中文、圖示等多元的字元做出符合程式碼目標的命名。第三個方法是將程式碼簡化、合併，用最少的程式碼直覺的教導 Swift 程式語言的觀念，並做出取捨來簡化教導的內容。以上三個方法讓初學者能輕易的看出重點而幫助學習。如圖 5-1～圖 5-10。

（二） 如果要讓 app 上架，使用 office 的軟體製作可能會有字體侵權問題，使用 macOS 的 Pages 文書軟體做 Swift 的視覺化資料再加以編輯製作 app 就可以避免字體侵權。

（三） 我們在實驗中使用的「設計 app 循環四階段法」能幫助我們聚焦在 app 目的、想法、適用對象，經過重新思索後更能堅持設計 app 的初衷，再規劃 app 的使用者介面、製作 app 的原型，最後評估 app 的成效。

（四） 在 Xcode 使用 view controller、Navigation controller、「Button」元件、「Image View」元件再搭配 segue 以及編輯好的視覺化圖檔，而設計出使用者只要經由簡單的點選動作，就可以輕鬆學習的 app，符合我們的期望。我們不只製作出 app，還利用 procreate 來製作 app icon。經由使用者在 Xcode 的模擬器使用 app 的回饋，發現我們原來的目的有達成，如果要讓 app 更完美，則需要把 app 加以遊戲化，這會讓學習 Swift 程式語言更有趣。讓 app 增加遊戲化是非常有挑戰性的，期待我們精熟 Swift 程式語言及 Xcode 的操作再加上一些創意後，能完成這個目標，讓 app 上架來造福更多對學習程式語言有興趣的人。

柒、參考資料

一、The Swift Programming Language(Swift 4.2)

iOS 的「書籍」app