當我們在圖 1-3 選擇「Command Line Tool」，按下 Next 按鈕後，畫面將如圖 1-5 所示，請在 Project Name 輸入 myFirst，並在 Organization Name 和 Organization Identifier 輸入一個名稱，這些都可以自行命名，最後在 Language 的選項選取「Swift」。

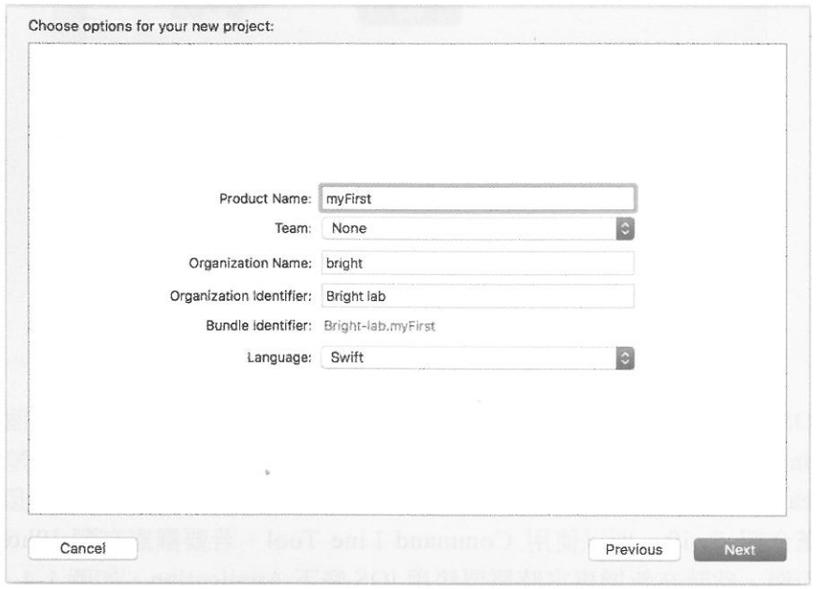


圖 1-5 專案名稱命名及編譯語言選擇

Xcode 8 目前仍提供 Objective-C 語言的編譯，但由於目前專案要介紹 Swift，因此 Language 的項目請選擇「Swift」。

按下 Next 按鈕後，請選擇專案欲儲存的位置，位置可以任你選擇，此範例儲存在文件下的「Swift program」資料夾，最後按下 Create 按鈕，如圖 1-6 所示。

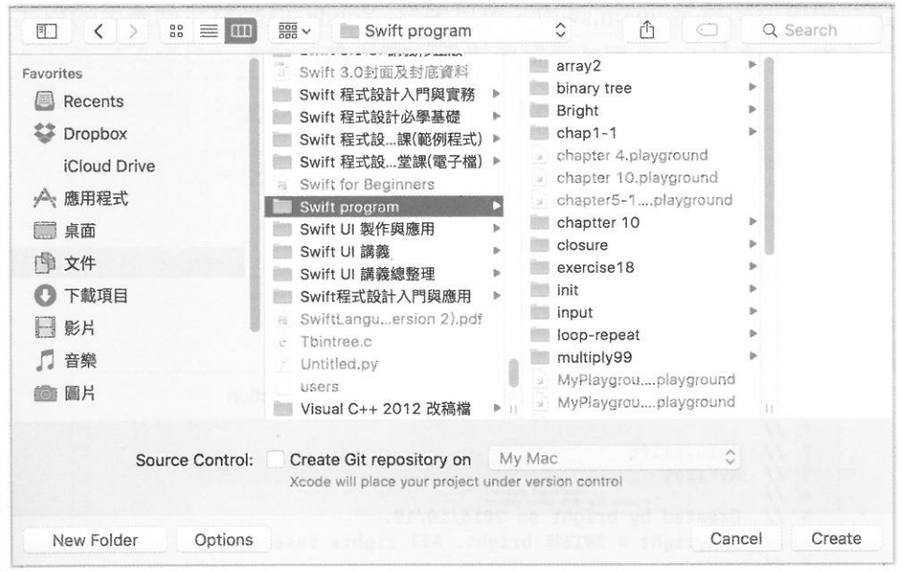


圖 1-6 專案儲存位置

完成後，將會出現如圖 1-7 的編輯與執行的畫面，請點選 myFirst 下的 main.swift。

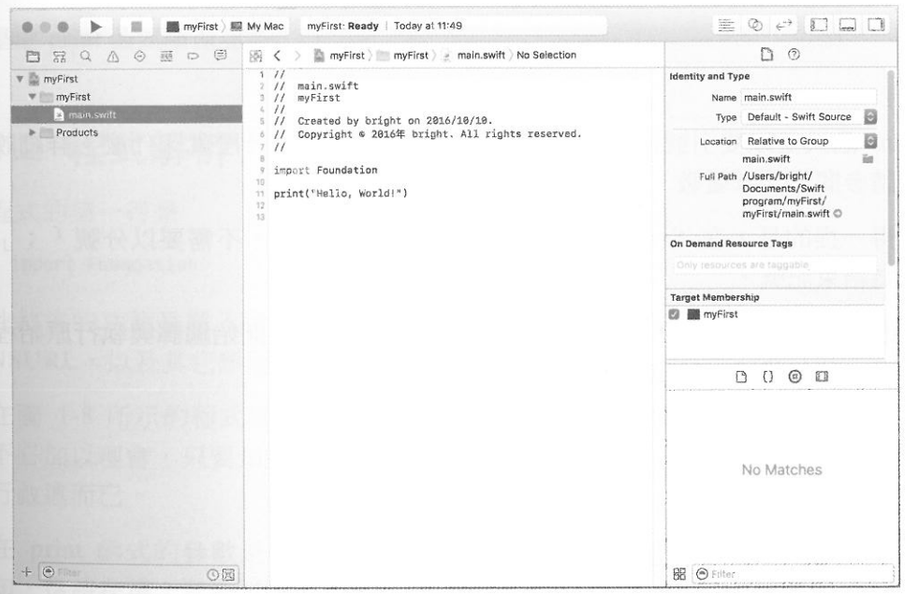


圖 1-7 編輯與執行的畫面

點選左側欄中的檔案 main.swift，可以看見 Xcode 自動幫我們產生的程式樣板。其中 .swift 為 Swift 程式語言所使用的副檔名。

接著我們做一點小小的修改，將下一行敘述

```
print(”Hello, world!”)
```

改為

```
print(”Learning Swift now!”)
```

如圖 1-8 所示：

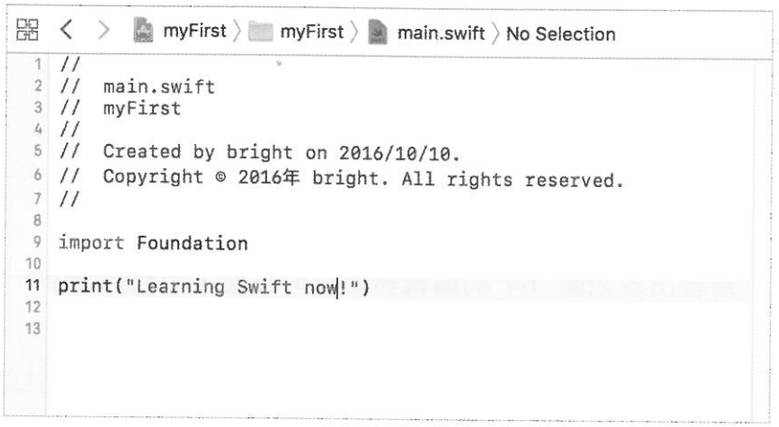


圖 1-8 修改後的程式

print 函式是以雙引號括起來的字串參數。字串中出現什麼就印什麼。詳細說明請參閱第 2 章變數、常數以及資料型態。

值得一提的是，在 Swift 語言中每一行程式碼的結尾，不需要以分號「；」作為結束記號。

修改完後，按下圖 1-7 左上角的「▶」按鈕，Xcode 會開始編譯與執行原始程式碼。若無錯誤，輸出結果如圖 1-9 所示：

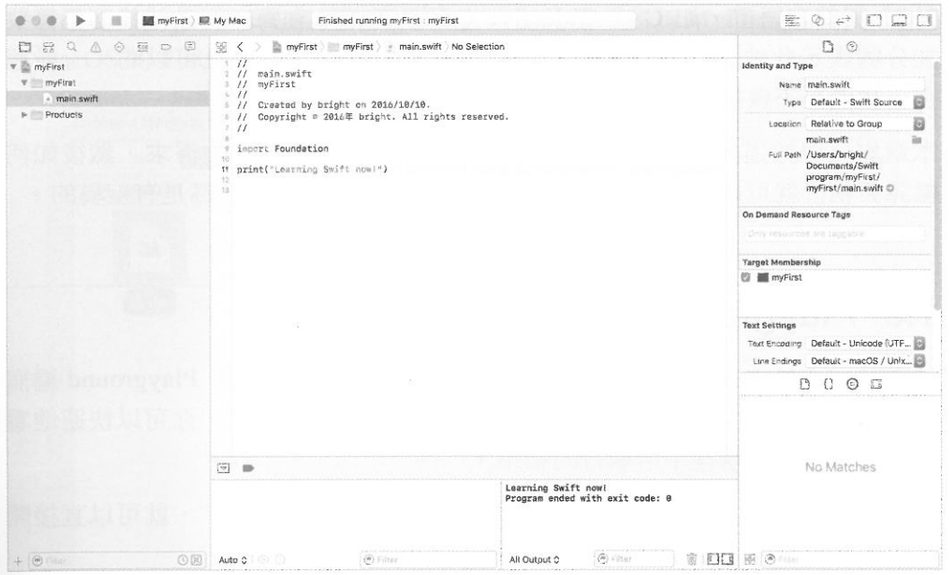


圖 1-9 輸出結果的畫面

在圖 1-9 下方的輸出結果欄位顯示

```
Learning Swift now!
```

是由 print 函式所產生。

1.2 程式解析

程式的第一行是

```
import Foundation
```

此敘述的功能是載入所有對 Swift 有效的 Foundation API，包括 NSDate、NSURL，以及其它類別的所有方法、屬性以及類目。

在圖 1-8 所示的程式碼，皆為系統自動產生，也就是上述所列的敘述您可以不必加以理會，只要知道其功能即可，我們只修改了第 11 行 print 函式這一行敘述而已。

在 print 函式的參數是一字串，置於此內的文字將被印出，除非有特殊的告知，這將在第 2 章再詳述之。

Swift 與其它語言，如 C 或 Objective C 不同的是，在每一行敘述後面不需要加分號表示此敘述已結束。不過你在每一敘述加上分號也是可以的。一般而言，我們是不會加的。

本章您只要知道如何建立一專案，從而修改程式以符合您的需求，然後如何編譯與執行就可以了。還有在程式中所使用英文字母，大小寫是有差異的。

1.3 Playground 介紹

Playground 是 Xcode 中自有的 Swift 程式碼開發環境。使用 Playground 編寫 Swift 程式碼，不需要編譯或執行一個要編譯的 Swift 程式，你可以快速地看到程式碼執行的過程中所執行的結果。

首先打開 Xcode 8，直接點擊 “Get started with a Playground”，就可以直接開啟一個 Playground 環境。如圖 1-10 所示：

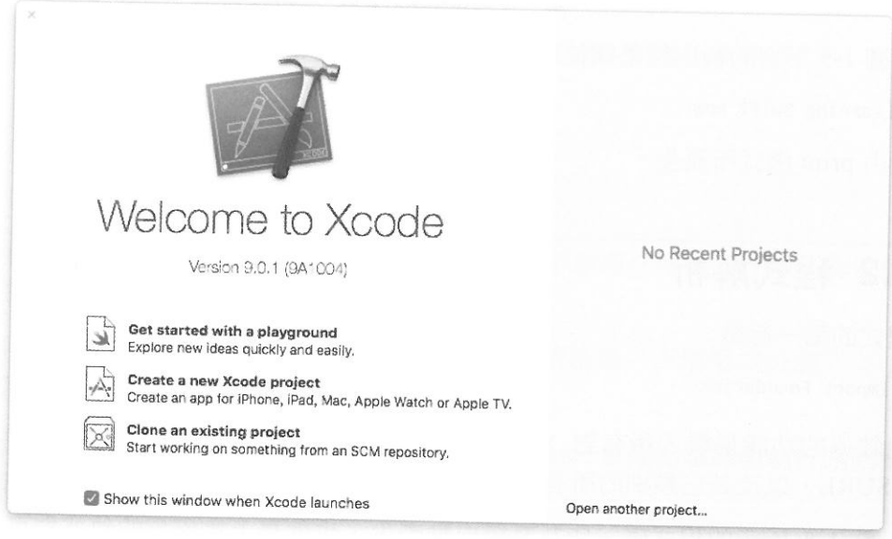


圖 1-10 選取畫面左邊的第一項

接著設定 Playground 的名稱和適用平台，在此是命名為 MyPlayground，並使用 macOS 的 Platform，然後按下 Next。如圖 1-11 所示：

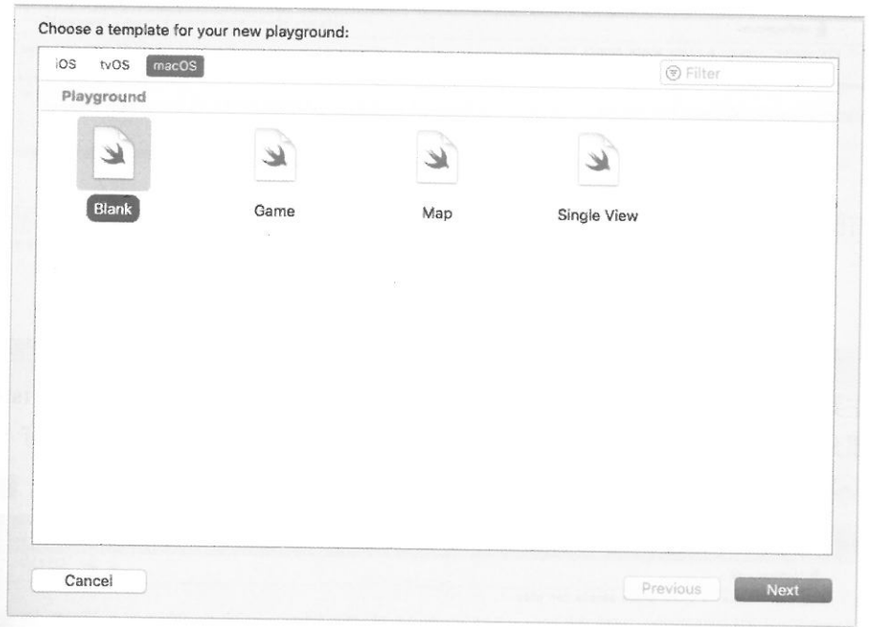


圖 1-11 設定 Playground 的名稱和適用平台

接下來選擇檔案要存放的位置，並按下 Create。完成後如圖 1-12 所示，

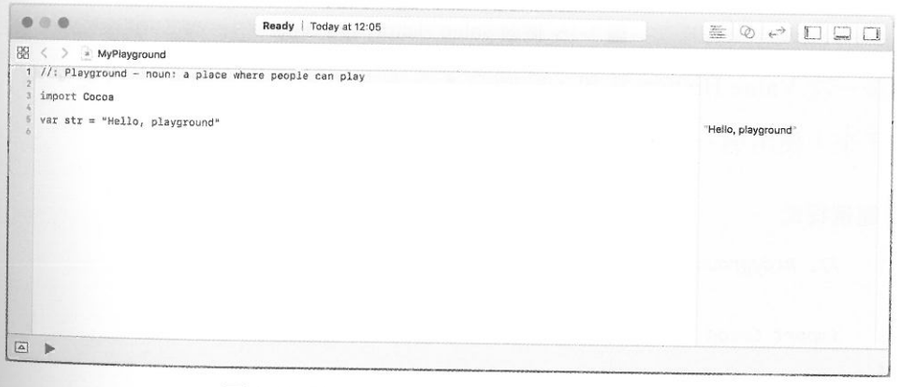


圖 1-12 Playground 的自動生成的範例程式碼

我們就可以開始使用 Playground 的環境來編寫 Swift 程式碼了。接著以自動生成的範例程式碼，來開始介紹 Playground 環境的內容。

首先將游標移到第 5 行，將會看到右側欄位即時顯示的部分（如圖 1-13）

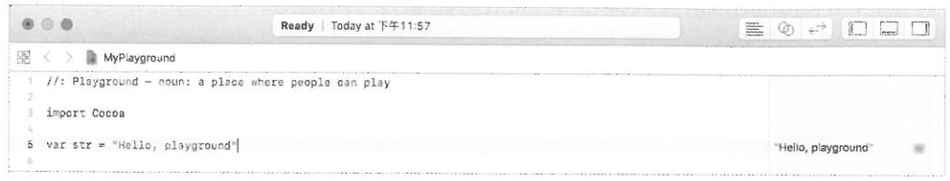


圖 1-13 Playground 的右邊側欄

在右側欄中的 "Hello, playground" 右邊有兩個按鈕，如圖 1-14 所示：

圖 1-14 Playground 的右邊側欄

由左到右分別是 Quick Look 和 Value History 按鈕，先來介紹 Value History 的部分。按下 Value History 按鈕後，輸出結果會出現在此程式碼的下方，如圖 1-15 第 5 行下的下方所顯示的字串。

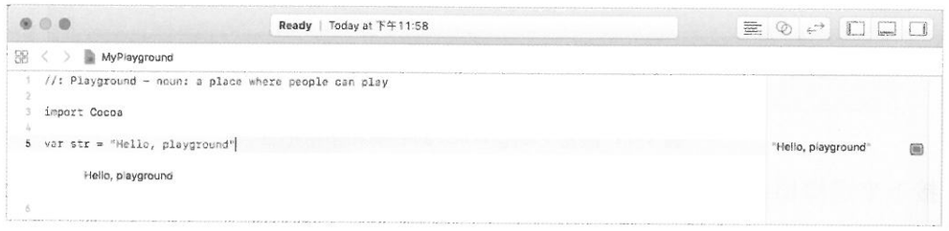


圖 1-15 開啟 Value History 後的結果

再按一次 Value History 按鈕，此時第 5 行下方所顯示的字串將會消失。

接下來，使用另一段範例程式碼來深入介紹。

範例程式

```
01  //: Playground - noun: a place where people can play
02
03  import Cocoa
04
05  var total = 0
06  for i in 1...100 {
07      total = total + i
08  }
09  print("1 加到 100 的總和: \(total)")
```

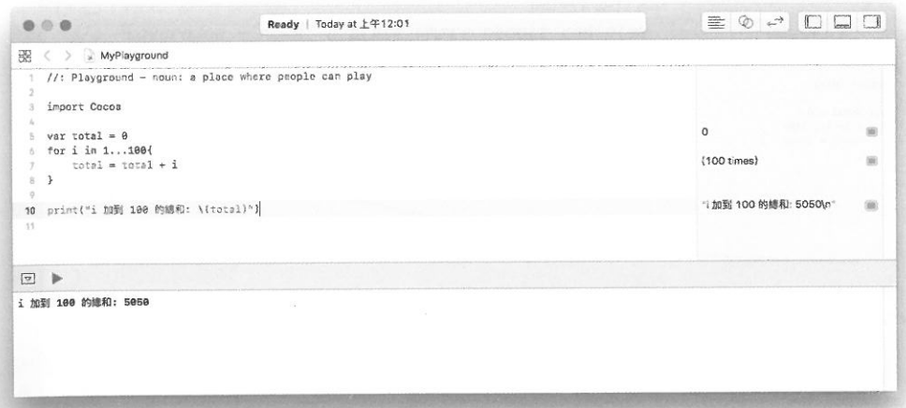



圖 1-16 計算 1 加到 100 總和的程式碼

在圖 1-16 右方的執行結果中，可以看到迴圈跑了 100 次，以及 1 加到 100 的總和: 5050。您也可以程式下方的「▶」按鈕來執行此程式，其結果將顯示於下方的區域。

接著點選圖 1-16 第 7 行右方區塊的 Value History 按鈕，在程式碼的下方將會看到結果，如圖 1-17 所示。

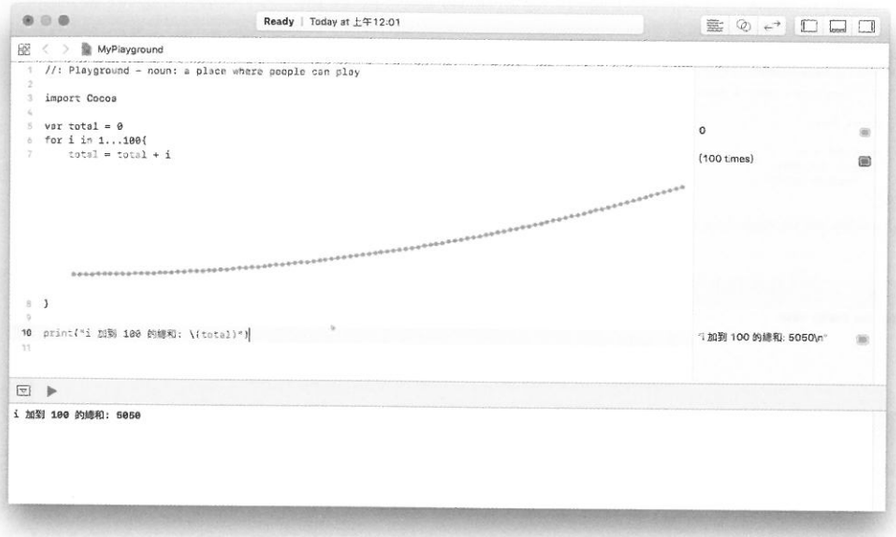


圖 1-17 按下 Value History 按鈕的結果

您可以看到 total 的 100 次變化，針對迴圈每一次的運行都會留下一個記錄，而這些變化會形成一個圖表，方便我們去看程式運行中的變化過程。

接著點選圖 1-16 第 7 行右方區塊的 Quick Look 按鈕，則在右側欄將會看到以下結果，如圖 1-18 所示。

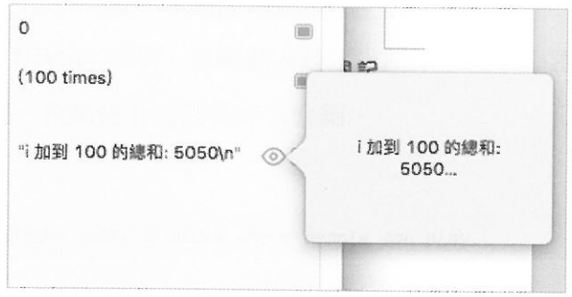


圖 1-18 按下 Quick Look 按鈕的結果

再舉另一段程式碼，來體會一下 Playground 的強大。

範例程式

```
01  //: Playground - noun: a place where people can play
02
03  import Cocoa
04
05  var sinCurve : Double
06  for i in 0..<100 {
07      sinCurve = sin(Double(i)/10)
08  }
```

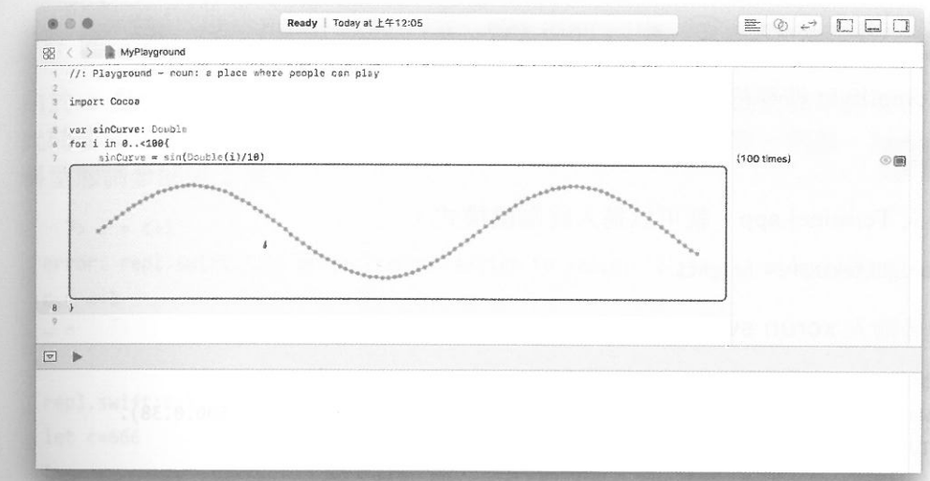


圖 1-19 sin 函式程式碼

在圖 1-19 中可以清楚看到，這是一段 sin 函式曲線的程式碼，如圖 1-20 所示。

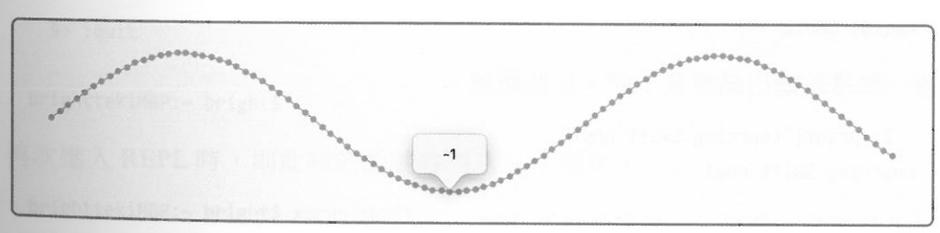


圖 1-20 sin 函數曲線圖表

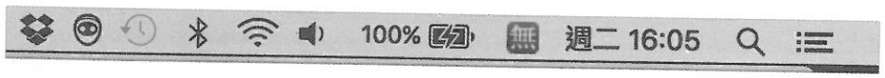
在圖 1-20 可以點擊不同位置，得知當時該變數的具體數值。

經由以上的介紹，相信您已經充分體會到 Playground 強大以及有趣的地方了吧！

1.4 REPL 介紹

Swift 還有一更快速的學習方法是使用讀取－運算－輸出 迴圈（Read-Eval-Print Loop, REPL）。我們來看看如何使用 REPL。

首先利用右上角



的 Spotlight 搜尋符號

輸入 Terminal.app，就可以進入終端機模式。

```
brighttekiMBP:~ bright$
```

接著輸入 **xcrun swift**，進入 Swift 的 REPL 模式，如下所示：

```
brighttekiMBP:~ bright$ xcrun swift
Welcome to Apple Swift version 3.0 (swiftlang-800.0.46.2 clang-800.0.38).
Type :help for assistance.
  1>
```

此時便可輸入敘述來執行。其中 1> 表示目前的進行狀況：例如要顯示 Hello, world，則輸入以下敘述：

```
  1> print("Hello, world")
Hello, world
```

每一敘述的輸出結果其下面。以此類推。

```
  2> print("Learning Swift now!")
Learning Swift now!
```

以下是宣告一變數 k，其初始值為 100，

```
  3> var k=100
k: Int = 100
```


接著將 k 加以印出。

```
  4> print(k)
100
```

若要在字串中印出變數 k 的值，則需要以 \(k) 方式撰寫，如下所示：

```
  5> print("k=\(k)")
k=100
```

當要宣告一常數名稱時，則以 let 代替 var，如以下宣告一常數名稱 c，其初始值為 666。

```
  6> let c = 666
c: Int = 666
```

由於 c 是常數名稱，所以不能再做加、減的運算，如以下的敘述是錯誤的。此時會顯示錯誤所在，而且也會告訴您如何修正。有關變數、常數，以及資料型態請參閱第 2 章。

```
  7> c = c+1
error: repl.swift:7:3: error: cannot assign to value: 'c' is a 'let' constant
c = c+1
~ ^

repl.swift:6:1: note: change 'let' to 'var' to make it mutable
let c=666
^~~
var
```

上式的訊息告訴您，錯在不能指定某值給常數名稱，所以要將 let 改為 var。

若要結束 REPL，則可以鍵入 :quit 或 :q，如下所示：

```
  9> :quit

brighttekiMBP:~ bright$
```

再次進入 REPL 時，則此時的編號會再次從 1 開始。

```
brighttekiMBP:~ bright$ xcrun swift
Welcome to Apple Swift version 3.0 (swiftlang-800.0.46.2 clang-800.0.38).
Type :help for assistance.
  1>
```

若要宣告一整數是陣列，則以下一敘述表示之。

```
  1> var dreamCar = ["Maserati", "Porsche", "BMW"]
dreamCar: [String] = 3 values {
  [0] = "Maserati"
  [1] = "Porsche"
  [2] = "BMW"
}
```

此表示有一陣列名為 dreamCar，它有三個元素，分別是 Maserati、Porsche，以及 BMW。所以 dreamCar[0] 是 Maserati， dreamCar[1] 是 Porsche，以及 dreamCar[2] 是 BMW，有關陣列請參閱第 6 章。然後以 for 迴圈加以印出，如下所示：

```
  2> for car in dreamCar {
  3.     print(car)
  4. }
Maserati
Porsche
BMW
```

有關迴圈敘述請參閱第 4 章。也可以輸入選擇敘述 if...else，此時將會等到您輸入到第 10 行的 } 才表示結束。如下所示：

```
  5> var score = 89
score: Int = 89
  6> if score >= 60 {
  7.     print("Pass")
  8. } else {
  9.     print("Fail")
 10. }
Pass
```

有關選擇敘述請參閱第 5 章。以 REPL 來學 Swift 是非常容易入門而且很快就可以上手，大家可以試試看。不過它對簡短的程式較方便，較長的程式可利用上述的 Playground，或建立 Project 來處理較佳。

自我練習題

1. 請建立一專案名稱 mySecond，將系統產生的程式加以修改，以輸出您的姓名、出生年月日、就讀的學校與科系、手機號碼、地址等等資訊。順便熟悉從撰寫一程式到編譯與執行的步驟。

2. 將 1-3 節所介紹的 sin 函式曲線程式碼改為 cos 函式曲線程式碼。看看它在 Playground 上的變化為何。順便熟悉在 Playground 上的環境。