Swift Introduction

Yung Chin, Yen

October 12, 2020

Outline

1 The Basics

Swift 是 2014 年由蘋果公司在其開發者大會 WWDC 上所發佈的程式語言

The Swift Programming Language 一書中提到[1]：

1. Swift 用來撰寫 iOS 及 Mac OSX 的程式。
2. Swift 吸收了 C 和 Objective-C 的優點，使其在設計更具彈性、容易，且撰寫時有更多樂趣。
3. Swift 是基於成熟而且倍受喜愛的 Cocoa 和 Cocoa Touch framework，意即 Swift 可以直接使用這些現有的框架
4. Swift 兼具編譯語言的性能，以及直譯式語言的特性。

2 Learning Resources

2.1 Web sites

- swiftdoc.org
- Swift 官方線上學習手冊: The Swift Programming Language
- Swift 编程语言
- SwiftUI by Example
- https://www.hackingwithswift.com/100/swiftui
- Swift Developer Blog
- 第一章 - Class, Struct and Enum 的抉擇
- Hacking with SWIFT
- Github#swift

[1][Swift] (1) - Swift 簡介

- Stack Overflow#swift

2.2 Videos

- Swift Programming Tutorial for Beginners (Full Tutorial)
- iOS & Swift Videos

3 Swift 常數與變數

3.1 宣告常數和變數

常數和變數必須在使用前宣告，用 let 來宣告常數，用 var 來宣告變數。下面的範例展示了如何用常數和變數來記錄使用者嘗試登錄的次數：

```swift
let maximumNumberOfLoginAttempts = 10
var currentLoginAttempt = 0
```

你可以在一行中宣告多個常數或者多個變數，用逗號隔開：

```swift
var x = 0.0, y = 0.0, z = 0.0
var population = 8_000_000
x = 3
y = 4
```

3.2 Swift v.s. C/C++

Swift 的變數宣告與 C/C++ 的差異之一在於宣告時不必要先宣告變數型別，例如，同樣要宣告一小數變數與整數變數，C++ 的做法為

```cpp
int x = 20;
double pi = 3.1416;
```

```
<stdin>:1:4: error: consecutive statements on a line must be separated by ';'
int x = 20;
   ^
   ;
```

```
<stdin>:2:7: error: consecutive statements on a line must be separated by ';'
double pi = 3.1416;
      ^
      ;
<stdin>:1:1: error: cannot find 'int' in scope
int x = 20;
^~~
<stdin>:1:5: error: cannot find 'x' in scope
int x = 20;
    ^
<stdin>:2:1: error: cannot find 'double' in scope
double pi = 3.1416;
^~~~~~
<stdin>:2:8: error: cannot find 'pi' in scope
double pi = 3.1416;
       ^~
```

而 Swift 的寫法為：

```swift
var x = 20
var pi = 3.1416
```

3.3 Swift is a type-safe language

We declared the variable numOfStudent and assigned 42 (第 1 行) as its initial value, Swift will assigned it the type integer. We can't change its type: it will always be an integer.

```swift
1 var numOfStudent = 42
2 numOfStudent = 30
3 numOfStudent = 25.8   // cannot assign value of type 'Double' to type 'Int'
4 numOfStudent = "fourty" // wrong
```

3.4 常數與變數的命名

我們可以使用喜歡的字元作為常數和變數名，包括 Unicode 字元：

```swift
let π = 3.14159
let 你好 = "你好世界"
let ⊠⊠ = "dogcow"
π = 3.14 // try to change the value of constant and find out what happen
```

但 Swift 的變數亦有其限制，包括：常數與變數名不能包含數學符號、箭頭、保留的（或者非法的）Unicode 碼位、連線與制表字元（box-drawing characters），也不能以數字開頭，但是可以在常數與變數名的其他地方包含數字。

3.5 變數與常數的輸出

```swift
var x = 20
var pi = 3.1416
print(x)
print(pi)
```

```
20
3.1416
```

3.6 變數與常數的基本規範

- 使用前需先宣告
- 不能重複宣告

```swift
var x = 30
let y = 10
// invalid redeclaration of variable
var x = 40
let y = 5
```

3.7 Stored variable v.s. Computed variable

1. computed variable

```swift
var x: Double = 4.3
var y: Double = 3.4
var z: Double {
    return x*y
}
print(z)
x = 10
y = 10
print(z) //自動重新計算z值
```

```
14.62
100.0
```

2. contentView 裡的 body 就是 computed variable

```
var body: som View {
    return Text("Hello world")
}
```

4 基本運算

4.1 指派運算子

指派運算 (a = b)，表示用 b 的值來初始化或更新 a 的值：

```
let b = 10
var a = 5
a = b
// a 現在等於 10

let decimalInteger = 17
let binaryInteger = 0b10001       // 17 in binary notation
let octalInteger = 0o21           // 17 in octal notation
let hexadecimalInteger = 0x11     // 17 in hexadecimal notation
```

4.2 數值運算子

1. Swift 支援所有數值型別基本運算：
   - 加法 (+)
   - 減法 (-)
   - 乘法 (*)
   - 除法 (/)
   - 餘數 (%)

```
print(1 + 2)       // 等於 3
print(5 - 3)       // 等於 2
print(2 * 3)       // 等於 6
print(10.0 / 2.5)  // 等於 4.0
print(10 / 3)      // 等於 1
print(10 & 3)      // 等於 1
```

```
3
2
6
4.0
3
2
```

2. 不同數值的運算

```
"exports both"body
```

3. 加法運算子也用於 String 的拼接：

```
"hello, " + "world"  // 等於 "hello, world"
```

```
<stdin>:1:11: warning: result of operator '+' is unused
"hello, " + "world"  // 等於 "hello, world"
```

4.3 複合指派運算子（Compound Assignment Operators）

```swift
var a = 1
a += 2 // a 現在是 3 (ref:+=)
```

上述程式第 2 行中的 a += 2 為 a = a + 2 的另一種寫法，其他運算子如-、*、/、% 皆可套用同樣的簡寫方式。

4.4 邏輯運算子

邏輯運算的操作物件是邏輯布林值。Swift 支援基於 C 語言的三個標準邏輯運算。

- 邏輯非（!a）
- 邏輯且（a && b）
- 邏輯或（a || b）

5 Property Observers

5.1 Set/Get

在以往的經驗中，對變數進行 assign 是一件直覺而簡單的事，例如：

```swift
var x: Int = 4
var y: Int = 5
var z: Int

z = x + y
print(z)
```

```
9
```

類似 C#，在 Swift 中，我們卻能透過 Set/Get 來介入 assign 的過程，例如：

```swift
1  var x: Int = 4
2  var y: Int = 5
3  var z: Int {
4      get { return x + y }
5      set { y = newValue - x }
6  }
7
8  print(z)
9  z = 30
10 print("x新值: \(x), y新值: \(y)")
```

```
9
x 新值: 4, y 新值: 26
```

如上例中，我們可以夾帶一些程式碼，在有人賦予 z 新值時 (z=?) 利用 set(第 5 行) 觸發這些程式碼；同理，也可以在有人讀取其值時觸發。而 newValue 代表被 assign 的新值。

又如下例，某學生的期中考三科成績就可以在輸入成績後自行判斷各科是否及格

```swift
class MidExam {
    var id: String = ""
    var chPass = false
    var ch: Int = 0 {
        didSet {
            if ch >= 60 {chPass = true}
        }
    }
    var maPass = false
    var ma: Int = 0 {
        didSet {
            if ma >= 60 {maPass = true}
        }
    }
}

let stsc = MidExam()
stsc.id = "202010101"
stsc.ch = 70
stsc.ma = 32

print(stsc.chPass)
print(stsc.maPass)
```

```
true
false
```

上述程式第 5 及 11 行中的 didSet 會在變數被 assign 新值時觸發執行，而 willSet 則是在變數即將被 assign 時觸發。

5.2 willSet/set/didSet

- 使用 set 時，因為變數的值正在被改變，所以有 newValue 來儲存新值；
- 使用 willSet 則變數尚未被改變；
- 使用 didSet 則是變數已被改變，所以有 oldValue

根據 apple 的說法，willSet 和 didSet 不會在 init 中被呼叫：The willSet and didSet observers of superclass properties are called when a property is set in a subclass initializer, after the superclass initializer has been called. They are not called while a class is setting its own properties, before the superclass initializer has been called.[2]

6 控制流程

6.1 Conditional Statement

The following examples were from docs.swift.org.

[2]Properties

1. if In its simplest form, the if statement has a single if condition. It executes a set of statements only if that condition is true.

   (a) single condition

   ```
   var temperatureInFahrenheit = 30
   if temperatureInFahrenheit <= 32 {
       print("It's very cold. Consider wearing a scarf.")
   }
   ```

   (b) combined conditions

   ```
   var mathScore = 91
   if mathScore <= 100 && mathScore >= 90 {
       print("Grade A")
   }
   ```

2. if-else The if statement can provide an alternative set of statements, known as an else clause, for situations when the if condition is false.

```
temperatureInFahrenheit = 40
if temperatureInFahrenheit <= 32 {
    print("It's very cold. Consider wearing a scarf.")
} else {
    print("It's not that cold. Wear a t-shirt.")
}
```

3. if-else if You can chain multiple if statements together to consider additional clauses.

```
temperatureInFahrenheit = 90
if temperatureInFahrenheit <= 32 {
    print("It's very cold. Consider wearing a scarf.")
} else if temperatureInFahrenheit >= 86 {
    print("It's really warm. Don't forget to wear sunscreen.")
} else {
    print("It's not that cold. Wear a t-shirt.")
}
```

4. switch In its simplest form, a switch statement compares a value against one or more values of the same type.

```
switch some value to consider {
case value 1:
    respond to value 1
case value 2,
     value 3:
```

```
    respond to value 2 or 3
default:
    otherwise, do something else
}
```

This example uses a switch statement to consider a single lowercase character called someCharacter:

```swift
let someCharacter: Character = "z"
switch someCharacter {
case "a":
    print("The first letter of the alphabet")
case "z":
    print("The last letter of the alphabet")
default:
    print("Some other character")
}
```

```swift
// 把分數轉成等第
// A(90-100), B(80-90), C(70-80), D(60-70), E(60-)
import Foundation

let str = readLine()    //從鍵盤讀入一字串, 存到str中
let choice = Int(str!)   //把str轉成整數, 存入score

switch choice! {
case 90..<101:
    print("A")
case 80..<90:
    print("B")
case 70..<80:
    print("C")
default:
    print("E")
}
```

6.2 For-In Loops

1. 語法

```swift
for counter in lower...upper{
    your codes
}
```

2. DEMO

```swift
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 = 10
var answer = 1
for _ in 1...power {
    answer *= base
}
print("\(base) to the power of \(power) is \(answer)")
```

```
3 to the power of 10 is 59049
```

6.3 While Loops

A while loop performs a set of statements until a condition becomes false. These kinds of loops are best used when the number of iterations is not known before the first iteration begins. Swift provides two kinds of while loops:

- while evaluates its condition at the start of each pass through the loop.
- repeat-while evaluates its condition at the end of each pass through the loop.

1. while A while loop starts by evaluating a single condition. If the condition is true, a set of statements is repeated until the condition becomes false.

   (a) 語法

   ```
   while condition {
       your codes here
   }
   ```

   (b) DEMO

   ```
   var counter = 5
   while counter > 0 {
       print("Hello, \(counter)")
   ```

```
    counter -= 1
}
```

```
Hello, 5
Hello, 4
Hello, 3
Hello, 2
Hello, 1
```

2. repeat ... while The other variation of the while loop, known as the repeat-while loop, performs a single pass through the loop block first, before considering the loop′ s condition. It then continues to repeat the loop until the condition is false.

   (a) 語法

   ```
   repeat {
       your codes here
   } while condition
   ```

   (b) DEMO

   ```
   var counter = 5
   repeat {
       print("Hello, \(counter)")
       counter -= 1
   } while counter > 0
   ```

   ```
   Hello, 5
   Hello, 4
   Hello, 3
   Hello, 2
   Hello, 1
   ```

7 集合型別

Swift provides three primary collection types, known as arrays, sets, and dictionaries, for storing collections of values. Arrays are ordered collections of values. Sets are unordered collections of unique values. Dictionaries are unordered collections of key-value associations.[3]

[3]The Swift Programming Language Swift 5.3

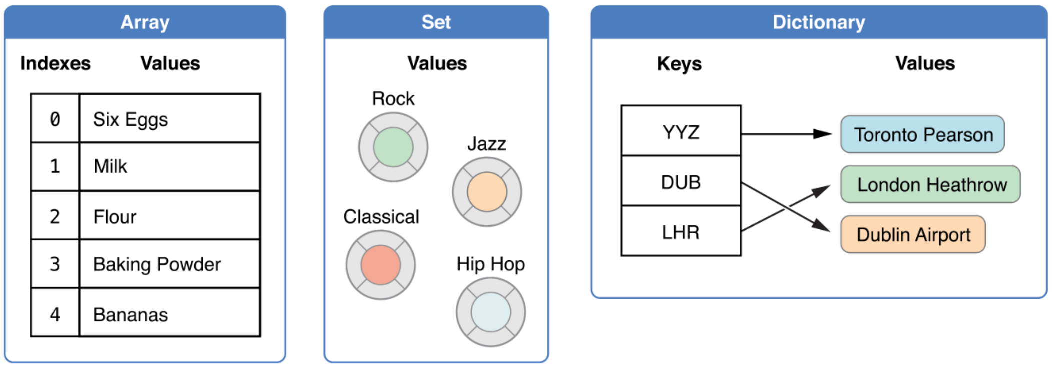


Figure 1: Collection Types of Swift

7.1 陣列（Arrays）

陣列使用有序列表儲存同一型別的多個值。相同的值可以多次出現在一個陣列的不同位置中。

1. 陣列建構語法 要建立一個空的整數陣列：

```
var EnglishScores = [Int]()
print(EnglishScores)
```

```
[]
```

下面這個範例創建了一個叫做 shoppingList 並且儲存字串的陣列：

```
var shoppingList: [String] = ["Eggs", "Milk"]
// shoppingList 已經被建構並且擁有兩個初始項。
```

2. 存取和修改陣列 我們可以通過陣列的方法和屬性來存取和修改陣列，或者下標語法。還可以

   - 使用陣列的唯讀屬性 count 來獲取陣列中的資料項數量
   - 使用布林項 isEmpty 來作為檢查 count 屬性的值是否為 0
   - 使用 append 方法在陣列後面添加新的資料項
   - +=
   - remove(at: index)

```
var shoppingList: [String]  = ["Eggs", "Milk"]
print(shoppingList.count)
print(shoppingList.isEmpty)
```

```
shoppingList += ["Bird"]  // Not += "Bird"
shoppingList.append("Fish")
print(shoppingList)
print(shoppingList[0])
shoppingList += ["Beer"]
print(shoppingList)
shoppingList.remove(at: 1)
print(shoppingList)
```

```
2
false
["Eggs", "Milk", "Bird", "Fish"]
Eggs
["Eggs", "Milk", "Bird", "Fish", "Beer"]
["Eggs", "Bird", "Fish", "Beer"]
```

3. 以 for 遍歷陣列

```
var shoppingList: [String]  = ["Eggs", "Milk", "Beer"]
for index in 0...2 {
    print(shoppingList[index])
}
for item in shoppingList {
    print(item)
}
```

```
Eggs
Milk
Beer
Eggs
Milk
Beer
```

4. 以 repeat 產生 array If you need to create an array of a specific size holding some default values, Swift has a built-in initializer called repeating:count:. You tell it what to repeat, and how often, and Swift will generate an array of that size.[4]

```
let nums1 = [Int](repeating: 0, count: 10)
let str1 = [String](repeating: "TNFSH", count: 5)
print(nums1)
print(str1)
```

[4]How to create an array by repeating an item

```swift

let nums2 = [[Int]](repeating: [Int](repeating: 6, count: 3), count: 2)
print(nums2)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
["TNFSH", "TNFSH", "TNFSH", "TNFSH", "TNFSH"]
[[6, 6, 6], [6, 6, 6]]
```

5. Modify array
   (a) map: 批次 update array 通常我們會使用到 map 函數的原因，主要是要將陣列的內容進行批次變更[5]，如：

```swift
let origin = ["James", "Vanessa", "Ruby", "Letranger"]
let lowercaseVersion = origin.map { $0.lowercased() }
print(lowercaseVersion)

let letterCounts = origin.map { $0.count }
print(letterCounts)

let nums = [0, 1, 2, 3, 4, 5]
let doubleNums = nums.map { (number) -> Int in
    number * 2
}
print(doubleNums)

let doubleStrings = nums.map { (number) -> String in
    "2 * \(number) = \(number * 2)"
}
print(doubleStrings)
```

```
["james", "vanessa", "ruby", "letranger"]
[5, 7, 4, 9]
[0, 2, 4, 6, 8, 10]
["2 * 0 = 0", "2 * 1 = 2", "2 * 2 = 4", "2 * 3 = 6", "2 * 4 = 8", "2 * 5 = 10"]
```

6. flatMap() flatMap 其實就是 flat + map 的結果。而 flatMap 最常是用來處理攤平某個陣列的狀況[5]

```swift
let nums = [2, 3, 4]
let mapped = nums.map { Array(repeating: $0, count: $0) }
print(mapped)
```

[5]Swift —Transforming an Array

```
4
5 let flatMapped = mapped.flatMap{$0}
6 print(flatMapped)
```

```
[[2, 2], [3, 3, 3], [4, 4, 4, 4]]
[2, 2, 3, 3, 3, 4, 4, 4, 4]
```

上述程式中第 5 行的效果與 Array(nums.map(transform).joined()) 相同。

7. compactMap compactMap 所映射出的元素結果都是非可選（non-nil）的值。因此，當你的希望映射結果的陣列是非可選類型的，那麼使用 compactMap 是一個適合的做法[5]。

```
let possibleNums = ["1", "2", "three", "///4///", "5"]
let mapped = possibleNums.map { str in Int(str) }
print(mapped)

let comMapped = possibleNums.compactMap { str in Int(str) }
print(comMapped)
```

```
[Optional(1), Optional(2), nil, nil, Optional(5)]
[1, 2, 5]
```

8. Array of Struct 基本的陣列提供了我們儲存大批資料的機制，然而，上述所示範的簡易型陣列彈性略嫌不足，如果我們想要開發一個圖書相關的 app，一開始我們至少需要這樣的陣列：

```
let book = ["X的悲劇", "地獄藍調","東方列車謀殺案","八百萬種死法","血字研究"]
```

然而，書本至少還要有作者吧，於是，我們可能又會加入作者的陣列:

```
let book = ["X的悲劇", "地獄藍調","東方列車謀殺案","八百萬種死法","血字研究"]
let author = ["艾勒里. 昆恩", "李查德", "阿嘉莎⊠克莉絲蒂","勞倫斯.卜洛克","柯南道爾"]
```

這種設計方式雖然可行，但是卻有幾個問題：

(a) 書名與作者存在不同陣列中，而且書名與作者並無直接的連結

(b) 當我們要再加入出版年份、出版社、售價. . . 等資訊時，就會發現需要太多的陣列了

Struct 就是一種十分適合用來儲存相關資料的型別，以上述書籍資料為例，我們可以先建立一個這樣的 Struct:

```
struct Book {
    var title: String
    var author: String
}
```

接下來，再建立一個上述 struct 的陣列，如此一來，不管書籍資料再如何複雜，我們都可以一個陣列來儲存：

```
struct Book {
    var title: String
    var author: String
}
var books = [
    Book(title:"X的悲劇", author: "艾勒里. 昆恩"),
    Book(title:"地獄藍調", author: "李查德"),
    Book(title:"東方列車謀殺案", author: "阿嘉莎☒克莉絲蒂"),
    Book(title:"八百萬種死法", author: "勞倫斯.卜洛克"),
    Book(title:"血字研究", author: "柯南道爾")
]
// 存取陣列中struct的方式
print(books[1].title)
print(books[1].author)
```

```
地獄藍調
李查德
```

7.2 集合（Sets）

集合（Set）是 Swift 集合類型（collection types）之一，集合用來存儲類型相同且沒有確定順序唯一的值。

1. 宣告

```
var letrers = Set<Character>()
```

2. 存取集合

```
var letters = Set<Character>()
print(letters.count)
letters.insert("a")
print(letters.isEmpty)
letters.insert("b")
letters.insert("c")
letters.insert("d")
letters.remove("c")
```

```swift
for item in letters {
    print("\(item)")
}
```

```
0
false
a
d
b
```

3. 交集、聯集

```swift
let oddDigits: Set = [1, 3, 5, 7, 9]
let evenDigits: Set = [0, 2, 4, 6, 8]
let primeNumbers: Set = [2, 3, 5, 7]

print(oddDigits.union(evenDigits))
print(oddDigits.union(evenDigits).sorted())
print(oddDigits.intersection(evenDigits).sorted())
print(oddDigits.subtracting(primeNumbers).sorted())
print(oddDigits.symmetricDifference(primeNumbers).sorted())
```

```
[1, 6, 9, 3, 4, 7, 8, 2, 5, 0]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[]
[1, 9]
[1, 2, 9]
```

4. 學習資源
   - Swift 中的集合 (Set)

7.3 字典（Dictionaries）

字典是一種儲存多個相同型別的值的容器。每個值（value）都關聯唯一的鍵（key），鍵作為字典中的這個值資料的識別符號。和陣列中的資料項不同，字典中的資料項並沒有具體順序。我們在需要通過識別符號（鍵）存取資料的時候使用字典，這種方法很大程度上和我們在現實世界中使用字典查字義的方法一樣[6]。

1. 建立字典

[6]《The Swŋift Prŋogrammŋing Laŋnguageŋ》正體中文版

```
var namesOfIntegers = [Int: String]()
// namesOfIntegers is an empty [Int: String] dictionary
namesOfIntegers[16] = "Sixteen"
print(namesOfIntegers)
```

```
[16: "Sixteen"]
```

一個字典字面量是一個定義擁有一個或者多個鍵值對的字典集合的簡單語句。一個鍵值對是一個 key 和一個 value 的結合體。在字典字面量中，每一個鍵值對的鍵和值都由冒號分割。這些鍵值對構成一個列表，其中這些鍵值對由方括號包含並且由逗號分割：

```
[key 1: value 1, key 2: value 2, key 3: value 3]
```

```
<stdin>:1:6: error: expected ',' separator
[key 1: value 1, key 2: value 2, key 3: value 3]
     ^
    ,
<stdin>:1:2: error: cannot find 'key' in scope
[key 1: value 1, key 2: value 2, key 3: value 3]
 ^~~
<stdin>:1:18: error: cannot find 'key' in scope
[key 1: value 1, key 2: value 2, key 3: value 3]
                 ^~~
<stdin>:1:34: error: cannot find 'key' in scope
[key 1: value 1, key 2: value 2, key 3: value 3]
                                 ^~~
```

2. 存取字典

```
var autherOfWriting: [String: String] = ["老人與海": "海明威", "異鄉人": "卡謬", "百年孤寂": "
    馬奎斯", "浪淘沙": "東方白"]
print(autherOfWriting)
print("一共有\(autherOfWriting.count)本書")
print(autherOfWriting.isEmpty)
autherOfWriting["鼠疫"] = "卡謬"
for (writing, author) in autherOfWriting {
    print("\(writing)的作者為\(author)")
}
```

```
["浪淘沙": "東方白", "老人與海": "海明威", "異鄉人": "卡謬", "百年孤寂": "馬奎斯"]
一共有4本書
```

```
false
浪淘沙的作者為東方白
老人與海的作者為海明威
異鄉人的作者為卡謬
鼠疫的作者為卡謬
百年孤寂的作者為馬奎斯
```

8 Functions

Functions are self-contained chunks of code that perform a specific task. You give a function a name that identifies what it does, and this name is used to "call" the function to perform its task when needed.

Swift's unified function syntax is flexible enough to express anything from a simple C-style function with no parameter names to a complex Objective-C-style method with names and argument labels for each parameter. Parameters can provide default values to simplify function calls and can be passed as in-out parameters, which modify a passed variable once the function has completed its execution.

8.1 Functions 的定義與呼叫

1. 語法

```
func name(arg1 param1: DataTyp, arg2 param2: DataType) -> returnValue DataType {
     your code here
     return returnValue
}
```

2. DEMO

```
"exports both"body
```

3. Functions 的參數與傳回值 Function 可以沒有參數，也可以有很多參數，可以沒有傳回值，也可以有很多傳回值 (這點與 C/C++ 不同)

   (a) 沒有參數

```
func sayHelloWorld() -> String {
    return "hello, world"
}
print(sayHelloWorld())
```

   (b) 有很多參數

```swift
func greet(person: String, alreadyGreeted: Bool) -> String {
    if alreadyGreeted {
        return "Hello, " + person + " again."
    } else {
        return "Hello, " + person
    }
}
print(greet(person: "Tim", alreadyGreeted: true))
// Prints "Hello again, Tim!"
```

function 的參數可以有兩種名稱：internal name 與 external name，internal 用於 function 內；external name 提供外部程式呼叫 function 使用。若 function 有一個以上的參數，可以省略第一個參數的名稱，如：

```swift
func greet(_ person: String, hiAgain alreadyGreeted: Bool) -> String {
    if alreadyGreeted {
        return "Hello, " + person + " again."
    } else {
        return "Hello, " + person
    }
}
print(greet("Vanessa", hiAgain: true))
print(greet("James", hiAgain: false))
// Prints "Hello again, Tim!"
```

(c) 沒有傳回值

```swift
func greet(person: String) {
    print("Hello, \(person)!")
}
greet(person: "Dave")
```

(d) 有很多傳回值

```swift
func myDiv(dividend: Int, divisor: Int) -> (quotient: Int, remainder: Int) {
    return (dividend / divisor, dividend % divisor)
}

let results = myDiv(dividend: 10, divisor: 3)
print("商：\(results.quotient), 餘數：\(results.remainder)")
```

```
商：3，餘數：1
```

9 Generics

當使用者 ‘‘不在乎’’ 其資料型別時使用，典型的例子為陣列，陣列可以儲存各式各樣型態的資料：Int, Double, String, Struct. . .

陣列的定義方式如下

```
strut Array<Element> {
    //....
    func append(_ element: Element) {...}
}
```

上述程式第 3 行 append 的參數 Element 即為一 “don’t care’’ 型別，代表 <Element> 裡可以填上 Int, String, Struct. . . 。其正式名稱為 Type Parameter。

10 Types

10.1 Varietities of Types

- struct
- class
- protocol
- “Dont’ Care” type (generics)
- enum
- functions

10.2 Functions as Types

如同 Int 或 Double，function 也可以是一種 type，如：

```
var foo: (Double) -> Void
func doSomething(what: () ->)
```

上述程式中第 1 行的 foo 為一變數，其型別為：輸入一小數變數、不傳回值，第 2 行 function 中的參數 what 的型別為一個沒有輸入、傳回 bool 的變數。

如下例：

```
 1  func square(operand: Double) -> Double {
 2      return operand * operand
 3  }
 4
 5  var operation = square
 6  let result1 = operation(4)
 7  print(result1)
 8
 9  operation = abs
10  let result2 = operation(-3)
11  print(result2)
```

程式中的第 5 行中，令 operation 變數之型別為 square 這個 function，於是 operation 就可以如 function 般進行計算；也可以令 operation 為內建的 function(如 abs)

1. Closures 在將 function 視為參數傳遞時，也可以把 function 內容直接寫出來，這種 inlined function 就稱為 closure

11 Struct, Class, and Enum

struct, class, and enum are fundamentally important concepts for every iOS developer. It′ s difficult to imagine common iOS apps without them. [7]

Class 和 Struct 是一種多功能且靈活的構造體，作為程式碼中的構造基礎。你可以使用與常數，變數和函數完全相同的語法來定義屬性和方法，以便為 Class 和 Struct 增加功能[8]。

[7]Demystifying Struct, Class, and Enum in Swift 5

[8]Day-15 Swift 語法 (11) - Class 與 Struct 的愛恨交織

strudt	class
Value type	Reference type
Copied when passed or assigned	Passed around via pointers
Copy on write	Automatically reference counted
Functional programming	Object-oriented programming
No inheritance	Inheritance (single)
“Free” init initalizes ALL vars	“Free” init initializes NO vars
Mutability must be explicitly stated	Always mutable
Your “go to” data structure	Used in specific circumstances
Everything you’ve seen so far is a struct	The ViewModel in MVVM is always a class

- 大多變數均為 struct(除了 View 之外)，如 arrays, dictionaries, Ints, Bools, Doubles,
- functional programming: foucs 在資料的 functionality
- object-oriented programming: focus 在將資料與其 functionality 封裝 (encapsulate) 入某個 object
- single inheritance: 只能繼承自一個 class
- 支援 multiple inheritance 的語言：C++, Common Lisp, Perl, Python, R, Raku, Tcl[9].

11.1 語法

```
struct SomeStructure {
    // structure definition goes here
}
class SomeClass {
    // class definition goes here
}
```

11.2 DEMO

1. 宣告

```
struct Resolution {
    var width = 0
    var height = 0
```

[9]Multiple inheritance

```
}
class VideoMode {
    var resolution = Resolution()
    var interlaced = false
    var frameRate = 0.0
    var name: String?
}
```

2. Init 用於建立 stuct 或 class，為內部變數進行初始化。

```
struct DiceGme {
    init(numberOfDices: Int){
        // create a dice with that many dices
    }
}
```

3. Structure and Class Instances The Resolution structure definition and the VideoMode class definition only describe what a Resolution or VideoMode will look like. They themselves don′ t describe a specific resolution or video mode. To do that, you need to create an instance of the structure or class.

```
let someResolution = Resolution()
let someVideoMode = VideoMode()
```

4. Accessing Properties You can access the properties of an instance using dot syntax. In dot syntax, you write the property name immediately after the instance name, separated by a period (.), without any spaces:

```
print("The width of someResolution is \(someResolution.width)")
// Prints "The width of someResolution is 0"
print("The width of someVideoMode is \(someVideoMode.resolution.width)")
// Prints "The width of someVideoMode is 0"
```

You can also use dot syntax to assign a new value to a variable property:

```
someVideoMode.resolution.width = 1280
print("The width of someVideoMode is now \(someVideoMode.resolution.width)")
// Prints "The width of someVideoMode is now 1280"
```

11.3 Structures

```
struct Birthday {
  var day: Int = 12
  var month: Int = 12
  var year: Double = 1999

  func myBirthday() {
  print("I've born in \(year).\(month).\(day)")
  }
}

struct Person {
  var firstName: String = "Abboskhon"
  var lastName: String = "Shukurullaev"
  var phoneNumber: String = 123456
  var emailAddress: String = "abbsh24@gmail.com"

  func myInfo() {
  print("My name is \(firstName) \(secondName). My phone number is \(phoneNumber) and email address
      is \(emailAddress)")
  }
}
```

1. Mutating methods

   Structs have also got so-called mutating methods that play a role in updating the property values of a structure within an instance method. [7]

```
struct Counter {
  var count: Int = 0

  mutating func increment() {
    count += 1
  }
  mutating func increment(by amount: Int) {
    count += amount
  }
  mutating func reset() {
    count = 0
  }
}
var counter = Counter()   //default is 0
counter.increment()       //becomes 1
counter.increment(by: 9)  //becomes 10
counter.reset()           //reset to 0
```

11.4 Classes

Classes and structures are very similar, and both can be used to define properties and methods. [7]

1. Inheritance & Override methods and properties

   The biggest difference that structs do not have is hierarchical relations. Classes can have parent classes, that are called superclass, and child classes, that are called subclasses. [7]

```swift
class Animals {
    func animals() {
        print("Animals are mainly of 3 types: Land Animals, Sea animals, Air Animals")
    }
}

class LandAnimals: Animals {
    override func animals() {
        print("Land animals are cats, sheeps, horses.")
    }
}

class Cat: LandAnimals {
    override func animals() {
        print("I am a cat and I am a land animal.")
    }
}
```

11.5 結構跟類的比較 (Comparing Structures and Classes) [10]

1. 在 Swift 中的結構與類有許多相同之處，兩者皆能：
   - 定義屬性來儲存值
   - 定義方法來提供功能
   - 定義下標來提供訪問他們用下標語法的值
   - 定義初始化器來設定他們的初始狀態
   - 可被擴展以擴展其功能，超越預設的實現
   - 符合協議以提供某種標準功能
2. 類具有的附加功能，但結構沒有：

[10] Day 10: [Swift] 結構和類 (Struct and Class)

- 繼承使一個類能夠繼承另一個的特性
- 類型轉換使我們可以在運行時檢查和解釋類實例的類型
- 反初始化器允許類的實例釋放它已分配的任何資源
- 引用計數允許對類實例的多個引用

3. 結構和列舉是值型別 (Structures and Enumerations Are Value Types) 值型別是一種其值在被賦值給變數或常數時被複製，或者在傳遞給函數時被複製。實際上，Swift 的整數、浮點數、布林值、字符串、數組和字典中的所有基本型別都是值型別，並且在幕後實現為結構。所有結構和列舉都是 Swift 中的值型別。這代表著我們創建的任何結構和列舉實例以及它們作為屬性的任何值類型在代碼中傳遞時始終會被複製。[10]

   (a) 範例

```swift
struct Resolution {
    var width = 0
    var height = 0
}

let hd = Resolution(width: 1920, height: 1080)
var cinema = hd

cinema.width = 2048

print("cinema is now \(cinema.width) pixels wide")
// Prints "cinema is now 2048 pixels wide"

print("hd is still \(hd.width) pixels wide")
// Prints "hd is still 1920 pixels wide"
```

   由上述例子可知，當 cinema 被賦予 hd 當前的值，儲存在 hd 的值被複製到新 cinema 的實例。最後結果兩個擁有相同值但完全不同的實例，所以當修改 $\text{cinema.width} = 2048$ 的時候，並不會影響儲存在 hd 中的 width。[10]

4. 類是參考型別 (Classes Are Reference Types) 不同於值型別，當參考型別被指定給一個變數或常數，或是傳遞進一個函數的時候，並不會被複製。參考型別非副本，是使用相同存在的實例。[10]

   (a) 範例

```swift
class VideoMode {
    var interlaced = false
    var frameRate = 0.0
```

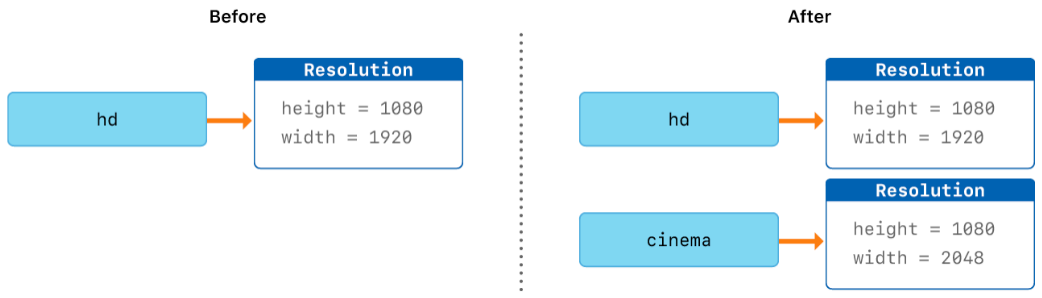


Figure 2: Struct value type

```
    var name: String?
}

let tenEighty = VideoMode()

tenEighty.interlaced = true
tenEighty.name = "1080i"
tenEighty.frameRate = 25.0


let alsoTenEighty = tenEighty
alsoTenEighty.frameRate = 30.0

print("The frameRate property of tenEighty is now \(tenEighty.frameRate)")
// Prints "The frameRate property of tenEighty is now 30.0"
```

由上述例子可知,當 alsoTenEighty 被指定為 tenEighty,並且修改 alsoTenEighty.frameRate = 30.0 時，也會更動到 tenEighty.frameRate 的值。[10]

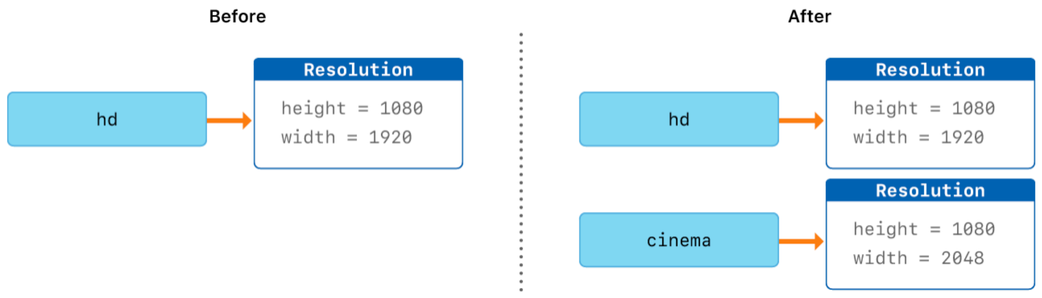


Figure 3: Class Reference type

11.6 with v.s. without struct: Why you need struct

1. 狀況一: without struct

```
//學生1資料
```

```
var st1-no = "S01"
var st1-name = "James"
var st1-english = 93
var st1-math = 90
//學生2資料
var st2-no = "S02"
var st2-name = "Vanessa"
var st2-english = 90
var st2-math = 99
```

2. 狀況二: with struct

```
struct Student {
    var no: String?
    var name: String
    var english = 0.0
    var math = 0.0
}

let st1 = Studetn(no: "S01", name: "James", english: 93, math: 92)
let st2 = Studetn()
st2.no = "S02"
st2.name = "Vanessa"
st2.english = 90
st2.math =  99
```

11.7 Inheritance

1. 使用 inheritance 的優點

   (a) 不用 inheritance

```
class Car {
    var topSpeed = 200
    func drive() {
        print("Driving at \(topSpeed)")
    }
}
class FutureCar {
    var topSpeed = 300
    func drive() {
        print("Driving at \(topSpeed)")
    }
    func autoDrive() {
        print("Auto-driving")
    }
}
let myRide = Car()
```

```
print(myRide.topSpeed)
myRide.drive()

let myNewRide = FutureCar()
myNewRide.drive()
myNewRide.autoDrive()
```

```
200
Driving at 200
Driving at 300
Auto-driving
```

(b) 使用 inheritance

```
class Car {
    var topSpeed = 200
    func drive() {
        print("Driving at \(topSpeed)")
    }
}
class FutureCar: Car {
    override func drive() {
        super.drive()
        print("and rockets boosting at 50")
    }

    func autoDrive() {
        print("Auto-driving")
    }
}
let myNewRide = FutureCar()
print(myNewRide.topSpeed)
myNewRide.drive()
```

```
200
Driving at 200
and rockets boosting at 50
```

11.8 Computer property

```
class Square {
    var edge = 0.0
    var area: Double {
        get {
            return edge*edge
        }
        set(newArea) {
            edge = newArea.squareRoot()
        }
    }
}

let square = Square()
square.edge = 10
print(square.area)
square.area = 36
print(square.edge)
```

```
100.0
6.0
```

11.9 Structures and Enumerations Are Value Types

```
struct Resolution {
  var width = 0
  var height = 0
}

let hd = Resolution(width: 1920, height: 1080)
var cinema = hd
```

When cinema was given the current value of hd, the values stored in hd were copied into the new cinema instance. The end result was two completely separate instances that contained the same numeric values. However, because they are separate instances, setting the width of cinema to 2048 doesn′ t affect the width stored in hd, as shown in the figure below[11]:

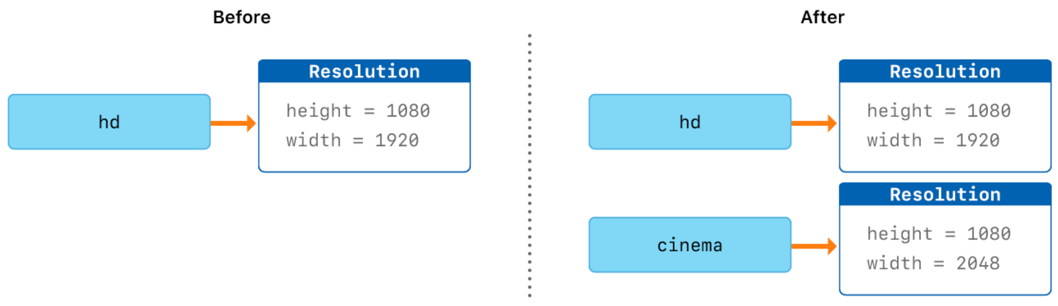


Figure 4: Value Types

[11]Structures and Classes

11.10 Classes Are Reference Types

```swift

struct Resolution {
  var width = 0
  var height = 0
}
class VideoMode {
    var resolution = Resolution()
    var interlaced = false
    var frameRate = 0.0
    var name: String?
}

let tenEighty = VideoMode()
tenEighty.resolution = hd
tenEighty.interlaced = true
tenEighty.name = "1080i"
tenEighty.frameRate = 25.0

let alsoTenEighty = tenEighty
alsoTenEighty.frameRate = 30.0
```

```
<stdin>:14:24: error: cannot find 'hd' in scope
tenEighty.resolution = hd
                       ^~
```

Because classes are reference types, tenEighty and alsoTenEighty actually both refer to the same VideoMode instance. Effectively, they are just two different names for the same single instance, as shown in the figure below[11]:

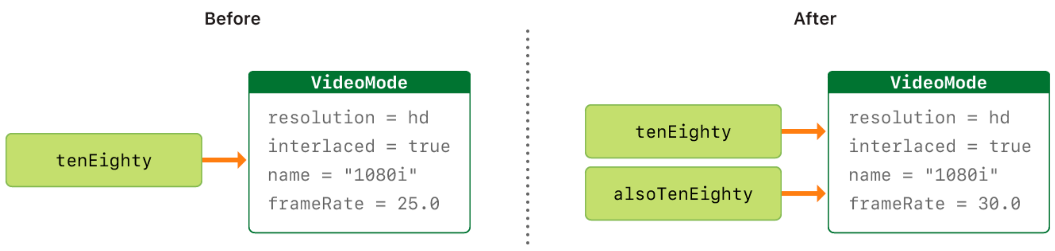


Figure 5: Value Types

11.11 Enumerations

Enumerations are different from structs and classes. Enum is a special Swift type that defines a common type for a group of related values. Then, enums can be interacted with other data types such as a switch, if statements, and others. [7]

從 Swift 3 開始，我們的 enum 內設定的情境字串要以小寫字母開頭。

```swift
//1 representation of enum
enum DownloadStatus {
    case downloading
    case finished
    case failed
    case cancelled
}

var currentStatus = DownloadStatus.downloading

switch currentStatus {
case .downloading:
    print("Downloading...")

case .finished:
    print("Just finished the download...")

case .failed:
    print("Failed to download the file...")

case .cancelled:
    print("The download is cancelled...")
}
```

此時，你可能會想，為什麼需要使用 enum 來定義多個情況，而不選擇宣告一個包含四個情境項目的 array，如下圖所示:

let downloadStatus = ["downloading" , "finished" , "failed" , "cancelled"]
let currentStatus = downloadStatus[0]

你可以這樣做沒錯,但是如此一來會有兩個缺點,首先,你可能會不知道 downloadStatus[0] 代表什麼，除非你引用 downloadStatus array，若是比較 downloadStatus[0] 與 DownloadStatus.downloading 這兩種表達方式，很明顯的是後者的可讀性比較高。

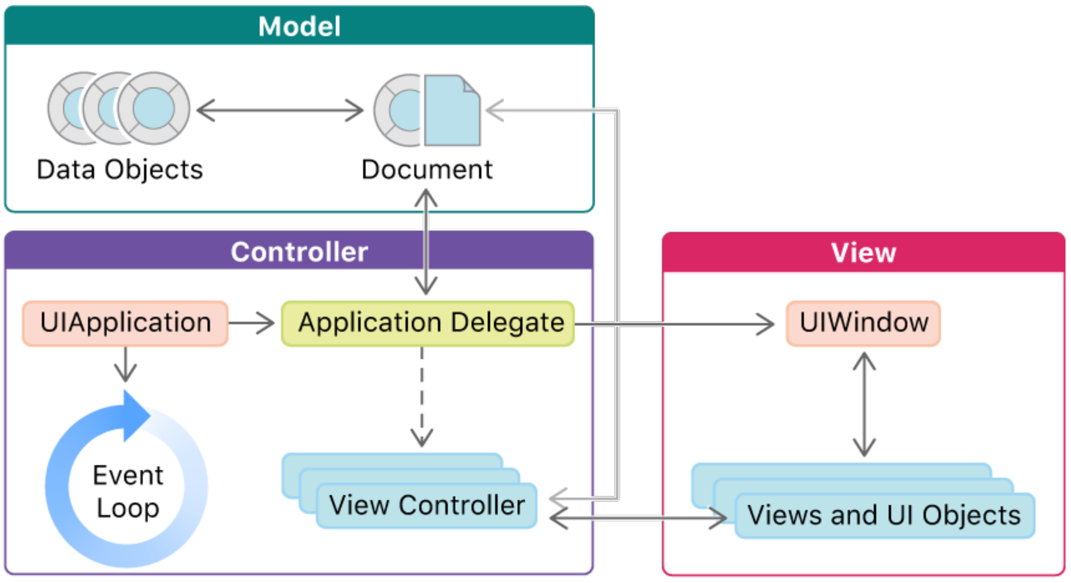
其次，因為 currentStatus 是 String 類型，變量可以被賦予任何字符串值，無法將它限制為 "downloading" , "finished" , "failed" 以及 "cancelled"，除非執行一些額外的驗證。反之，如果使用 enum，我們可以將 myDirection 限制在.downloading、.finished、.failed 或.cancelled 等四種情境之一，不會出現其他意料之外的情況。[12]

[12]精通 Swift：列舉、閉包、泛型、Protocols 和高階函數

11.12 結構和類之間的選擇 (Choosing Between Structures and Classes)

總的來說，用 class 來定義資料物件的話，就好像是在用雲端共享文件一樣：每個人的螢幕上都會有一份文件可以編輯，但這個文件並沒有存在電腦裡，而是跟雲端的版本連線，所以所有的變動都是直接在雲端版本上更新的。好處是方便，壞處是誰修改了甚麼東西經理不會知道（class 本身沒有帳號功能!）。

用 struct 的話，則是像傳統的離線文件檔案一樣。一開始文件只有經理有，而如果他想要讓手下小美去修改文件的話，他就需要拷貝一份檔案給小美。小美修改完檔案後，必須把它交還給經理，然後經理再決定要不要用修改過的檔案取代原本的文件。[13]



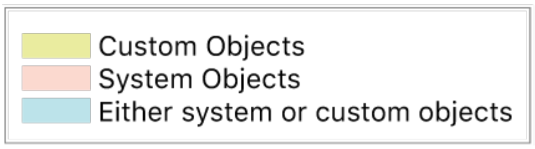


Figure 6: MVC

- 在 MVC 架構中，Model 包含了 Data Objects 與 Document，其中 Documents 包含所有的資料管理者元件 (如 Core Data 的 NSManagedObjectContext，或是任何負責下載、上傳資料物件的網路層元件，如自訂的 NetworkManager)。

- Document 最好是用 class 來定義，因為它肩負了許多溝通的工作。Data Objects 則是資料的代表，要用 class 或 struct 來定義都可以。

- 用 class 定義 Data Objects 的話，任何變動只要執行一次就可以了，因為它的實體只會有一個。然而，用 struct 定義的話，則需要將變動手動套用到文件所管理的那份實體，好讓整個 app 都

[13]Swift Class vs Struct：設計 Model 時，該用 Struct 還是 Class 呢？

能使用最新的資料。這雖然寫起來較為囉唆，卻讓閱讀與維護更為簡單 (可以確定是在哪裡被變更資料)[13]。

1. 選擇原則
   - Choose Structures by Default
   - Use Classes When You Need Objective-C Interoperability
   - Use Classes When You Need to Control Identity
   - Use Structures When You Don't Control Identity

12 協定