

資料科學入門 Python基礎實作

授課老師：林彥廷



課程簡介

- 巨量資料(Big Data)時代來臨，如何有效處理資料已成為各行業重要技能，而Python為目前巨量資料處理最廣為使用之程式語言，本課程以基礎實作讓學生學習Python程式語言，培養資料科學人才所需基礎能力。

課程內容

- Python開發環境操作
- 資料型態與變數宣告操作
- 判斷式與迴圈操作
- 方法定義與呼叫
- 實務程式設計與演練

下載及安裝Python軟體

- 到python.org下載軟體

The screenshot shows the Python.org homepage. At the top is a dark navigation bar with links: Python, PSF, Docs, PyPI, Jobs, and Community. Below this is a blue header with the Python logo and a search bar. A secondary blue bar contains links: About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area is split into two columns. The left column has a dark background with Python code examples for list comprehensions and the enumerate function. The right column has a blue background with the heading 'Compound Data Types' and a paragraph explaining lists, with a link to 'More about lists in Python 3'. At the bottom, a blue bar contains the text: 'Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)'.

Python

PSF

Docs

PyPI

Jobs

Community

python™

Search

GO

Socialize

About

Downloads

Documentation

Community

Success Stories

News

Events

```
# Python 3: List comprehensions
>>> fruits = ['Banana', 'Apple', 'Lime']
>>> loud_fruits = [fruit.upper() for fruit in fruits]
>>> print(loud_fruits)
['BANANA', 'APPLE', 'LIME']

# List and the enumerate function
>>> list(enumerate(fruits))
[(0, 'Banana'), (1, 'Apple'), (2, 'Lime')]
```

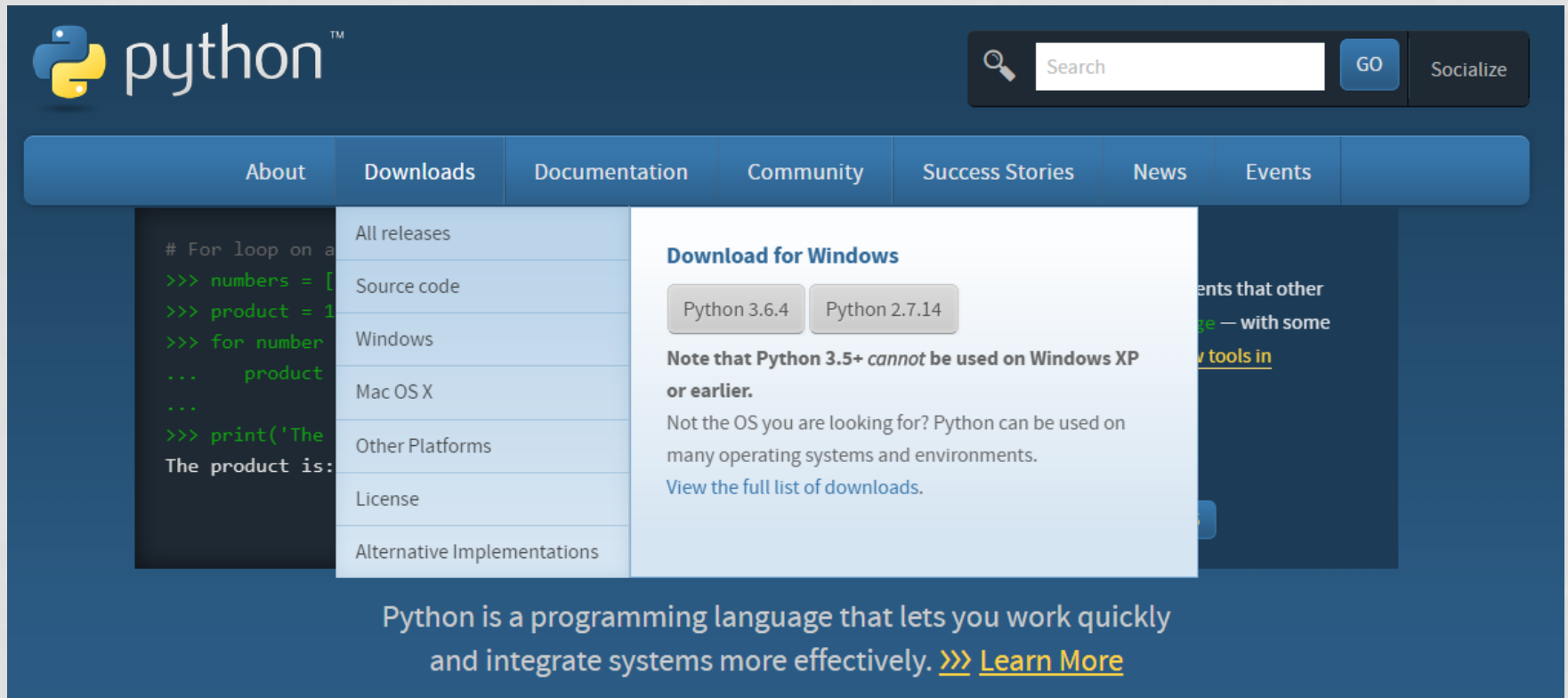
Compound Data Types

Lists (known as arrays in other languages) are one of the compound data types that Python understands. Lists can be indexed, sliced and manipulated with other built-in functions. [More about lists in Python 3](#)

1 2 3 4 5

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)

到python.org下載軟體



The screenshot shows the Python.org homepage with a dark blue header. The Python logo is on the left, and a search bar with a magnifying glass icon and a 'GO' button is on the right. Below the header is a navigation bar with links: About, Downloads, Documentation, Community, Success Stories, News, and Events. The 'Downloads' link is highlighted, and a dropdown menu is open, listing: All releases, Source code, Windows, Mac OS X, Other Platforms, License, and Alternative Implementations. The 'Windows' option is selected, leading to a 'Download for Windows' section. This section features two buttons: 'Python 3.6.4' and 'Python 2.7.14'. Below these buttons, a note states: 'Note that Python 3.5+ cannot be used on Windows XP or earlier.' This is followed by text: 'Not the OS you are looking for? Python can be used on many operating systems and environments. View the full list of downloads.' At the bottom of the page, a large blue banner contains the text: 'Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)'.

python™

Search GO Socialize

About Downloads Documentation Community Success Stories News Events

For loop on a
>>> numbers = [
>>> product = 1
>>> for number
... product
...
>>> print('The
The product is:

All releases
Source code
Windows
Mac OS X
Other Platforms
License
Alternative Implementations

Download for Windows

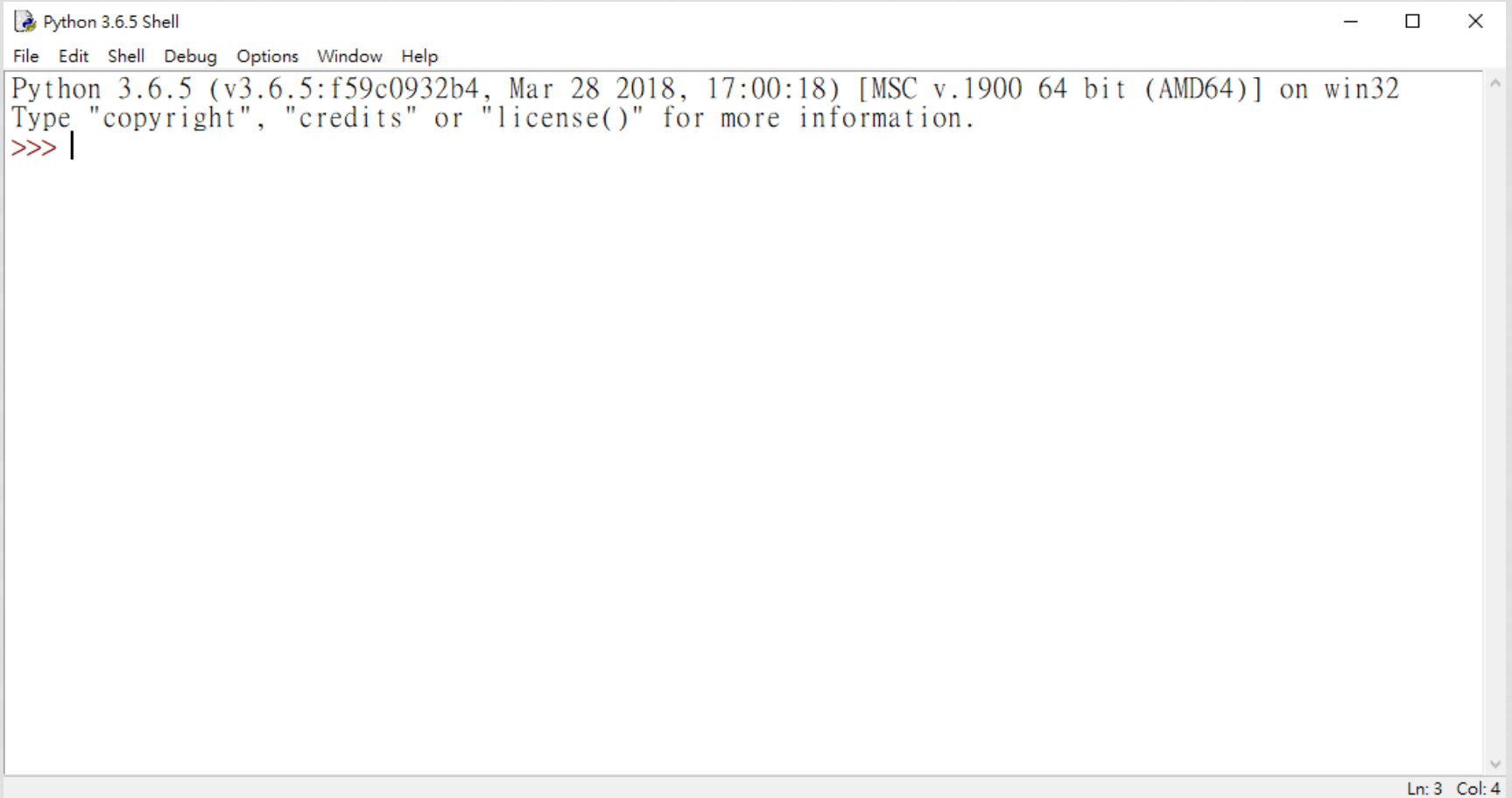
Python 3.6.4 Python 2.7.14

Note that Python 3.5+ cannot be used on Windows XP or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.
[View the full list of downloads.](#)

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)

Python IDLE開發環境



The image shows a screenshot of the Python 3.6.5 Shell window. The window has a title bar that says "Python 3.6.5 Shell" and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with the following options: File, Edit, Shell, Debug, Options, Window, and Help. The main area of the window displays the following text:

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

The status bar at the bottom right of the window shows "Ln: 3 Col: 4".

課程內容

- Python開發環境操作
- 資料型態與變數宣告操作
- 判斷式與迴圈操作
- 方法定義與呼叫
- 實務程式設計與演練

常用語法1

- `print()`
 - 將`print()`中的值輸出到螢幕上
- 例如
 - `Print('how old are you?')`
 - `x = 'how old are you?'`
 - `print(x)`

常用語法2

- `int()` `str()` `float()`
 - 將資料型態轉換為整數、字串、浮點數
- 例如
 - `x = 'this year is '`
 - `y = 2018`
 - `print(x + str(y))`

常用語法3

- `input()`
 - 使用者畫面會停留，並請使用者輸入值(Value)，所輸入value的資料型態會是字串(String)
- 例如
 - `input("how old are you?")`

常用語法4

- `.format()`
 - 結合`print()`使用，能夠自動將資料型態轉換為字串輸出至使用者畫面
- 例如
 - `Print('you are {} years old.'.format(x))`

資料型態與變數宣告操作

- Value 值
 - One of the basic units of data, like a number or string, that a program manipulates.
 - 程式最基本的執行單位，可能是數字或字元(串)
- 例如
 - 1
 - 'a'

資料型態與變數宣告操作

- Data Type 資料型態
 - Integer整數：int
 - Floating-point浮點數：float
 - String字串：string or str
- 例如
 - 1
 - 1.2
 - 'how are you'
 - 可使用type()來確認資料型態

資料型態與變數宣告操作

- Variable 變數
 - A name that refers to a value.
 - 用來儲存value的名稱
- 例如
 - $x = 1$
 - $y = \text{'Bob'}$

資料型態與變數宣告操作

- Expression 表示式
 - A combination of variables, operators, and values that represents a single result value.
 - 透過變數、值及運算子的組合，能夠將結果表示為單一value
- 例如
 - $x = 1$
 - $y = 2$
 - $z = x + y$

資料型態與變數宣告操作

- 資料型態轉換
 - 在進行程式設計時，值或變數的資料型態需一致才能夠進行運算
- 例如
 - $x = 1$
 - $y = '2'$
 - $z = x + \text{int}(y)$
 - $z = \text{str}(x) + y$

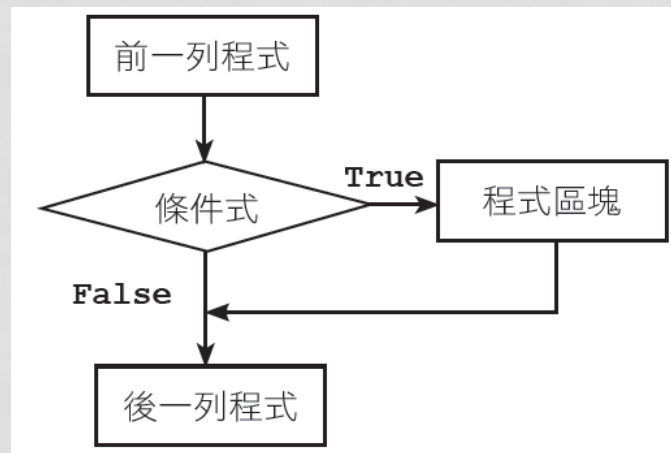
判斷式

- 單向判斷式

- 「if…」為單向判斷式，是 if 指令中最簡單的型態，語法為：

```
if (條件式):  
    程式區塊
```

- 以下是單向判斷式流程控制的流程圖：



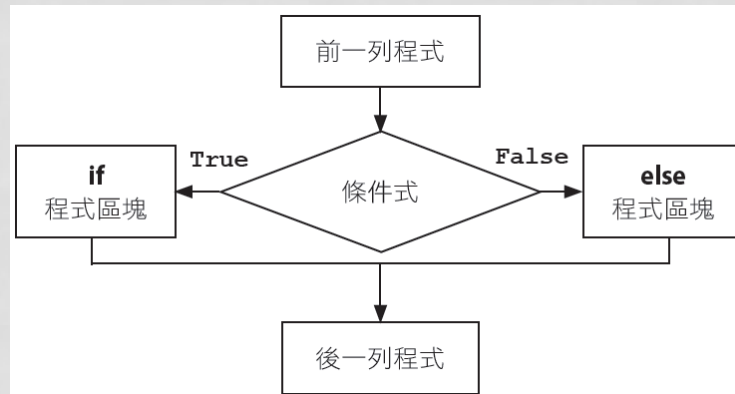
判斷式

- 雙向判斷式

- 「if...else...」為雙向判斷式，語法為：

```
if (條件式):  
    程式區塊一  
else:  
    程式區塊二
```

- 以下是雙向判斷式流程控制的流程圖：



判斷式

- 多向判斷式

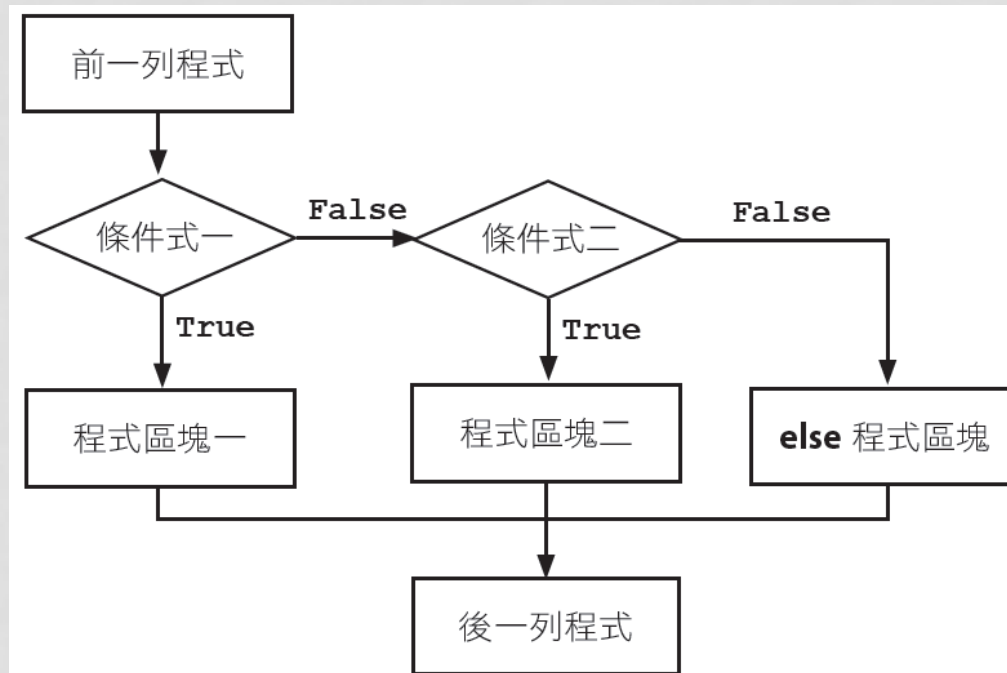
- 「if...elif...else」為多向判斷式，語法為：

```
if (條件式一) :  
    程式區塊一  
elif (條件式二):  
    程式區塊二  
elif (條件式三):  
    .....  
[else:]  
    程式區塊else
```

判斷式

- 多向判斷式

- 多向判斷式流程控制的流程圖 (以設定兩個條件式為例)：



判斷式

- 多重條件判斷
 - 如果同時有2個以上的條件須判斷，可以在if()判斷式中結合and 或 or來進行條件判斷
 - 例如
 - $a = 10$
 - $b = 6$
 - If($a > 11$ and $b < 7$)
 - Print('pass')

迴圈

- 專門用來處理重複事件的命令稱為「迴圈」
- Python 迴圈命令有2個：
 - for 迴圈用於執行固定次數的迴圈
 - while 迴圈用於執行次數不固定的迴圈。

range 函式

- 迴圈最常使用整數循序數列，例如「1,2,3,……」，每個數列的內容稱為數列的元素，range 函式的功能就是建立整數循序數列。

range 函式的語法

- range 函式單一參數
 - range 函式使用 1 個參數的語法為：

```
數列變數 = range(整數值)
```

- 例如：

```
list1 = range(5)    #數列為0,1,2,3,4
```

- 可以用 `print(list(list1))` 來看數列結果

range 函式的語法

- range 函式二個參數

- range 函式包含2個參數的語法為：

```
數列變數 = range(起始值, 終止值)
```

- 例如：

```
list2 = range(3, 8)    #數列為3,4,5,6,7
```

- 起始值及終止值皆可為負整數，例如：

```
list3 = range(-2, 4)   #數列為-2,-1,0,1,2,3
```

range 函式的語法

- range 函式三個參數
 - range 函式包含3 個參數的語法為：

數列變數 = range(起始值, 終止值, 間隔值)

- 例如：

```
list4 = range(3, 8, 1)    #數列為3,4,5,6,7
```

```
list5 = range(3, 8, 2)    #數列為3,5,7 ,數列每次增加2
```

- 間隔值也可為負整數，此時起始值必須大於終止值

```
list6 = range(8, 3, -1)   #數列為8,7,6,5,4 ,數列每次遞減1
```

for 迴圈

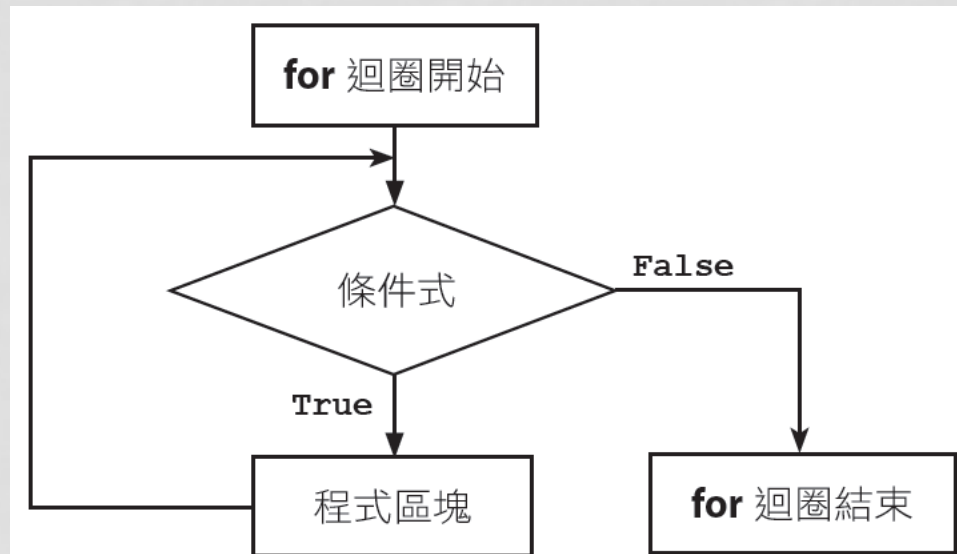
- for迴圈語法

```
for 變數 in 數列:  
    程式區塊
```

- 程式範例

1	for n in range(3):	#產生 0,1,2 的數列
2	print(n, end=",")	#執行結果為:0,1,2,

- for迴圈流程



break指令

- 迴圈執行時，如果要在中途結束迴圈執行，可以使用break指令強制離開迴圈
- 例如

```
for i in range(1,11):  
    if(i==6):  
        break  
    print(i, end=",")    #執行結果:1,2,3,4,5,
```

continue指令

- continue指令是在迴圈執行中途暫時停住不往下執行，而跳到迴圈起始處繼續執行
- 例如

```
for i in range(1,11):  
    if(i==6):  
        continue  
    print(i, end=",")    #執行結果:1,2,3,4,5,7,8,9,10,
```

while 迴圈

- while迴圈通常使用於沒有固定次數的情況
- 基本語法結構為：

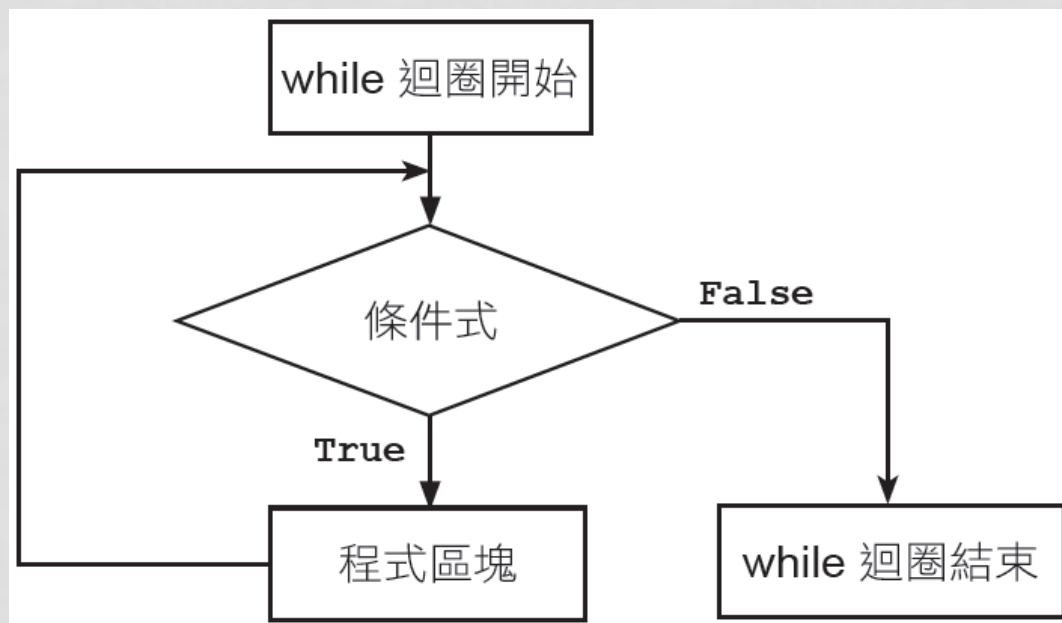
```
while (條件式) :  
    程式區塊
```

- 例如

```
1 total = n = 0  
2 while(n < 10):  
3     n += 1  
4     total += n  
5 print(total)    #1+2+.....+10=55
```

while 迴圈

- while 迴圈的流程如下：



while 迴圈

- 在使用 while 迴圈時要特別留意，必須設定條件判斷的中止條件，以便可以停止迴圈的執行，否則會陷入無窮迴圈的窘境。例如：

```
1 total = n = 0
2 while(n < 10):
3     total += n
4 print(total)
```


while 迴圈

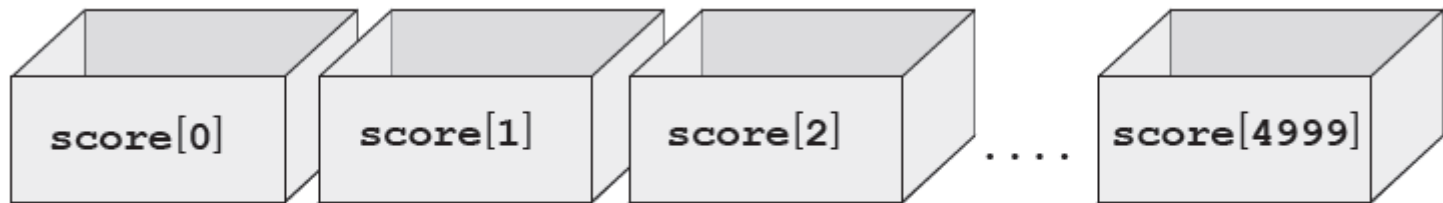
- 數學上定義「 $n! = 1 * 2 * 3 * \dots * n$ 」，所以 $1! = 1$ 、 $2! = 1 * 2 = 2$ 、 $3! = 1 * 2 * 3 = 6$ ，請利用 while 設計這個程式。當使用者輸入一個正整數 n 後，程式就會顯示由 $1 * 2 * 3 * \dots * n$ 的乘積
- 例如
 - 使用者輸入5
 - 系統輸出 $5! = 120$

課程內容

- Python開發環境操作
- 資料型態與變數宣告操作
- 判斷式與迴圈操作
- 方法定義與呼叫
- 實務程式設計與演練

串列 (List)

- 串列 (又稱為「清單」或「列表」)，與其他語言的「陣列 (Array)」相同，其功能與變數相類似，是提供儲存資料的記憶體空間。



▲ 串列元素配置

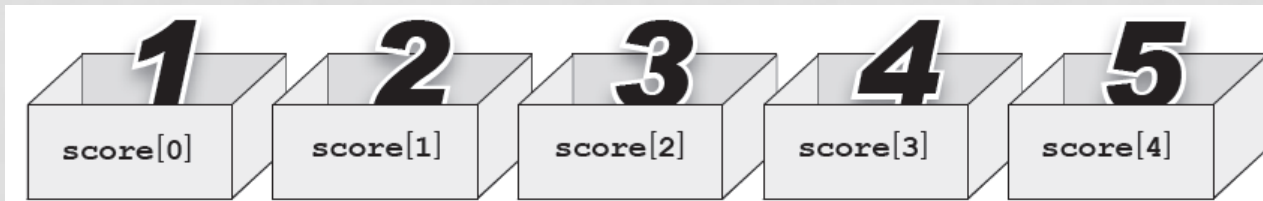
串列宣告

- 一維串列

- 一維串列的宣告方式是將元素置於中括號 ([]) 中，每個元素之間以逗號分隔，語法為：

串列名稱 = [元素 1, 元素 2, ……]

- 例如：宣告 score 串列，其元素內容為 [1, 2, 3, 4, 5]。



串列宣告

- 一維串列

- 串列中各個元素資料型態可以相同，也可以不同，例如：

<code>list1 = [1, 2, 3, 4, 5]</code>	<code># 元素皆為整數</code>
<code>list2 = ["香蕉", "蘋果", "橘子"]</code>	<code># 元素皆為字串</code>
<code>list3 = [1, "香蕉", True]</code>	<code># 包含不同資料型態元素</code>

串列宣告

- 空串列

- 例如：

```
list4=[]
```

- 多維串列宣告

- 例如下面是二維串列的範例，其串列元素是帳號、密碼組成的串列：

```
list5=[["joe","1234"],["mary","abcd"], ["david","5678"]]  
print(list5[1])      #["mary","abcd"]  
print(list5[1][1])   #abcd
```

串列元素的存取

- 讀取串列元素
 - 讀取串列元素的語法為：

串列名稱 [索引]

- 例如：
- 取得前面 list1 串列之中索引為 0 (第 1 個元素) 的元素內容，得到結果為 1。

```
list1 = [1, 2, 3, 4, 5]
```

```
print(list1[0])    #1
```

串列元素的存取

- 注意索引值是從 0 開始計數：第一個元素索引值為 0，第二個元素索引值為 1，依此類推。索引值不可超出串列的範圍，否則執行時會產生「list index out of range」錯誤。例如：

```
list4 = ["香蕉", "蘋果", "橘子"]  
print(list4[1])    # 蘋果  
print(list4[2])    # 橘子  
print(list4[3])    # 錯誤，索引值超過範圍
```


串列元素的存取

- 索引值可以是負值，表示由串列的最後向前取出，「-1」表示最後一個元素，「-2」表示倒數第二個元素，依此類推。同理，負數索引值不可超出串列的範圍，否則執行時會產生錯誤。例如：

<code>list4 = ["香蕉", "蘋果", "橘子"]</code>	
<code>print(list4[-1])</code>	# 橘子
<code>print(list4[-3])</code>	# 香蕉
<code>print(list4[-4])</code>	# 錯誤，索引值超過範圍

串列元素的存取

- 改變串列元素

- 語法為：

串列名稱 [索引] = 元素內容

- 如：將 list1 串列中索引為 0 (第 1 個元素) 的元素內容，由 1 改變為 9。

```
list1 = [1, 2, 3, 4, 5]
print(list1[0])    # 1
list1[0]=9         # 更改為 9
print(list1[0])    # 9
```

使用 for ... 迴圈讀取串列

- 使用 for 變數 in 串列讀取串列
 - 使用 for 迴圈可以讀取串列的元素，它相當於其他語言的 for ~ each，其基本語法結構為：

```
for 變數 in 串列：  
    程式區塊
```

- 以實例解說：

```
1 list1 = ["香蕉", "蘋果", "橘子"]  
2 for s in list1:  
3     print(s, end=",")    # 執行結果為：香蕉, 蘋果, 橘子,
```

使用 for ... 迴圈讀取串列

- 取得串列長度
 - 迴圈中 range() 函式的範圍通常會利用 len() 函式計算串列的長度。例如：計算 scores 串列的長度，顯示結果為 3。

```
scores = [85, 79, 93]  
print(len(scores)) # 3
```

- 以 for in range 迴圈讀取串列

```
scores = [85, 79, 93]  
for i in range(len(scores)):  
    print(scores[i])
```

串列搜尋與計次

- `index()` 搜尋

- 語法：

索引值 = 串列名稱 . `index` (串列元素)

- 例如：

```
list1 = ["香蕉","蘋果","橘子"]
```

```
n = list1.index("蘋果")    #n=1
```

```
m = list1.index("梨子")    #ValueError: '梨子' is not in list
```

串列搜尋與計次

- `count()` 計算次數

- 語法：

次數 = 串列名稱 .count (串列元素)

- 例如：

```
list1 = ["香蕉","蘋果","橘子"]
```

```
n = list1.count("橘子")    #n=1
```

```
m = list1.count("梨子")    #m=0
```

串列元素新增和刪除

- 增加串列元素

- append() 方法

- 語法：

串列名稱.append(元素值)

- 例如：

- 在 list1 串列最後面增加一個串列元素「金榜」。

```
list1 = [1,2,3,4,5,6]
```

```
list1.append("金榜")    #list1=[1,2,3,4,5,6,'金榜']
```

```
print(list1[6])        #金榜
```

```
print(len(list1))      #7
```

串列元素新增和刪除

- 增加串列元素

- append() 方法

- 語法：

```
串列名稱 .append( 元素值 )
```

- 例如：

- 新增一個空串列，並append()一個新元素於其中

```
scores = []  
scores.append(90)
```


串列元素新增和刪除

- 增加串列元素

- insert() 方法

- 語法：

串列名稱.insert(索引值, 串列元素)

- 例如：

- 在 list1 串列索引 3 的位置插入一個串列元素「紅榜」。

```
list1 = [1,2,3,4,5,6]
```

```
list1.insert(3,"紅榜")    #list1=[1,2,3,"紅榜",4,5,6]
```

```
print(list1[3])           # 紅榜
```

```
print(len(list1))         #7
```

串列元素新增和刪除

- 增加串列元素

- insert() 方法

- 如果索引值大於或等於串列元素個數，將如同append()方法一樣將串列元素加在最後面。
- 索引值也可以是負值，表示由串列的最後向前推算，「-1」表示最後一個元素，「-2」表示倒數第二個元素，依此類推。

- 例如：在 list1 串列索引第 -1、12 的位置插入串列元素。

```
list1 = [1,2,3,4,5,6]
```

```
list1.insert(-1, "愛") #list1=[1, 2, 3, 4, 5, '愛', 6]
```

```
list1.insert(12, "台灣") #list1=[1, 2, 3, 4, 5, '愛', 6, '台灣']
```

```
print(list1) # [1, 2, 3, 4, 5, '愛', 6, '台灣']
```

```
print(len(list1)) #8
```

串列元素新增和刪除

- 刪除串列元素

- del 語法

- 刪除串列單一元素語法(以索引值為基礎)：

```
del 串列名稱 (n1)
```

- 刪除串列指定範圍元素語法(以索引值為基礎)：

```
del 串列名稱 (n1:n2[:n3])
```

串列元素新增和刪除

- 刪除串列元素
 - del 語法
 - 例如：

```
list1 = [1,2,3,4,5,6]
```

```
del list1[1]
```

```
print(list1) #[1,3,4,5,6]
```

```
list2=[1,2,3,4,5,6]
```

```
del list2[1:5:2] # 刪除索引第 1、3 的串列元素
```

```
print(list2) #[1,3,5,6]
```

串列元素新增和刪除

- 刪除串列元素

- remove()方法：

- 用於移除列表中某個元素的第一個匹配元素：

串列名稱.remove(obj)

- 例如：

```
>>> list2 = ['1', '2', '3', '4']
```

```
>>> list2.remove('3')
```

```
>>> print(list2)
```

```
['1', '2', '4']
```

```
>>> list2=[1,2,3,4]
```

```
>>> list2.remove(4)
```

```
>>> print(list2)
```

```
[1, 2, 3]
```

串列元素排序

- 串列元素由小到大排序

- sort() 方法：

- 語法：

串列名稱 . sort ()

- 例如：將 list1 串列由小到大排序。

```
list1=[3,2,1,5]  #[3, 2, 1, 5]
```

```
list1.sort()
```

```
print(list1)  #[1, 2, 3, 5]
```

串列元素排序

- 反轉串列元素順序

- reverse()方法：

- 語法：

串列名稱.reverse()

- 例如：將 list1 串列元素順序反轉。

```
list1=[3,2,1,5]  #[3, 2, 1, 5]
```

```
list1.reverse()
```

```
print(list1)  #[5, 1, 2, 3]
```

串列元素排序

- 串列元素由大到小排序
 - `sort() + reverse()`方法：
 - 語法：

```
串列名稱.sort()  
串列名稱.reverse()
```
 - 例如：將 `list1` 串列由大到小排序。

```
list1=[3,2,1,5] #[3, 2, 1, 5]  
list1.sort()  
print(list1) #[1, 2, 3, 5]  
list1.reverse()  
print(list1) #[5, 3, 2, 1]
```


串列元素排序

- 串列元素大到小排序並儲存於新串列中

- 語法：

```
串列名稱 2=sorted( 串列名稱 1,reverse=True)
```

- 例如：將 list1 串列由大到小排序，並儲存在 list2 串列。

```
list1=[3,2,1,5] #[3, 2, 1, 5]
```

```
list2=sorted(list1,reverse=True)
```

```
print(list2)      #[5, 3, 2, 1]
```

```
print(list1)      #[3, 2, 1, 5]   # 原串列不變
```

自訂函式

- 在一個較大型的程式中，通常會將具有特定功能或經常重複使用的程式，撰寫成獨立的小單元，稱為「函式」，並賦予函式一個名稱，當程式需要時就可以呼叫該函式執行。
- 使用函式的程式設計方式具有下列的好處：
 - 將大程式切割後由多人撰寫，有利於團隊分工，可縮短程式開發的時間。
 - 可縮短程式的長度，程式碼也可重複使用，當再開發類似功能的產品時，只需稍微修改即可以套用。
 - 程式可讀性高，易於除錯和維護。

自訂函式

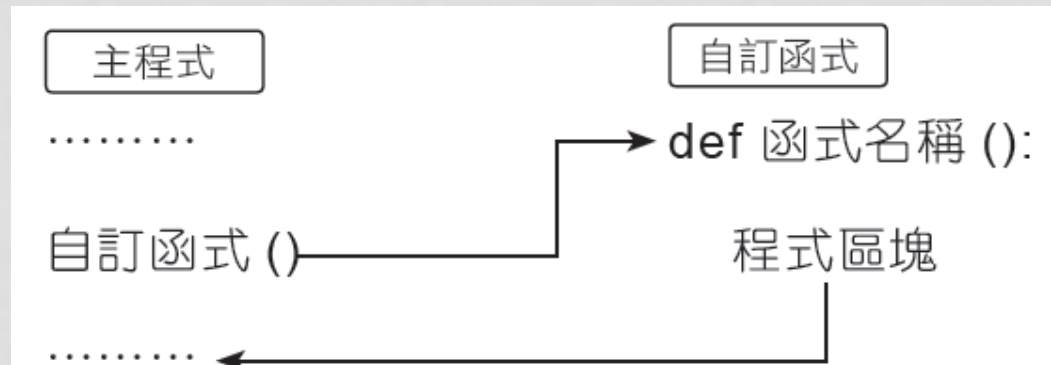
- Python 是以 def 命令建立函式，不但可以傳送多個參數給函式，執行完函式後也可返回多個回傳值。自行建立函式的語法為：

```
def 函式名稱 ([ 參數 1, 參數 2, …… ]):  
    程式區塊  
    [return 回傳值 1, 回傳值 2, ……]
```

自訂函式

- 函式建立後並不會執行，必須在主程式中呼叫函式，才會執行函式，呼叫函式的語法為：

[變數 =] 函式名稱 ([參數])



自訂函式

```
def sayhello():  
    print("welcome!!")  
  
sayhello()    #welcome!!
```

```
def sayhello(name):  
    print("welcome!! " + name)  
  
user_name = input("please input your name: ")  
sayhello(user_name)    #welcome!! Ricky
```

```
def sayhello(name):  
    hello_string = "welcome!! " + name  
    return hello_string  
  
user_name = input("please input your name: ")  
hello_result = sayhello(user_name)  
print(hello_result)    #welcome!! Ricky
```

自訂函式

```
def 函式名稱 ([ 參數 1, 參數 2, ..... ]):  
    程式區塊  
  
    [return 回傳值 1, 回傳值 2, .....]
```

- 參數 (參數 1, 參數 2,) : 參數可以傳送一個或多個，也可以不傳送參數。參數是用來接收由呼叫函式傳遞進來的資料，如果有多個參數，則參數之間必須用逗號「,」分開。
- 回傳值 (回傳值 1, 回傳值 2,) : 回傳值可以是一個或多個，也可以沒有回傳值。回傳值是執行完函式後傳回主程式的資料，若有多個回傳值，則回傳值之間必須用逗號「,」分開，主程式則要有多個變數來接收回傳值

自訂函式

- 如果函式有傳回值，可以使用變數來儲存返回值，特別注意若函式有多個傳回值，必須使用相同數量的變數來儲存返回值，變數之間以逗號「,」分開，如果參數的數量較多，常會搞錯參數順序而導致錯誤結果，呼叫函式時可以輸入參數名稱，此種方式與參數順序無關，可以減少錯誤。不過輸入參數名稱方式會多輸入不少文字，降低建立程式效率。

跨檔案呼叫函式

```
import pyFileName  
pyFileName.functionName()
```

```
import Week_2_function  
result = Week_2_function.sayhello('rickylin')  
print(result)
```


跨檔案呼叫函式

- 使用別名

```
import pyFileName as nickName  
pyFileName.functionName()
```

```
import Week_2_function as aa  
result = aa.sayhello('rickylin')  
print(result)
```

自訂類別

```
class className:  
    def functionName():  
        statement...
```

```
class A:  
    def sayhello():  
        print("welcome!!")  
    def sayname(name):  
        hello_string = "welcome!! " + name  
        return hello_string
```

跨檔案呼叫類別

```
from pyFileName import className  
className.functionName()
```

```
from Week_2_class import A  
A.sayhello()  
result = A.sayname('ricky')
```

跨檔案呼叫類別

- 使用別名

```
from pyFileName import className as nickName  
nickName.functionName()
```

```
from Week_2_class import A as aa  
aa.sayhello()  
result = aa.sayname('ricky')
```

THE END

ytlin@mail.nptu.edu.tw