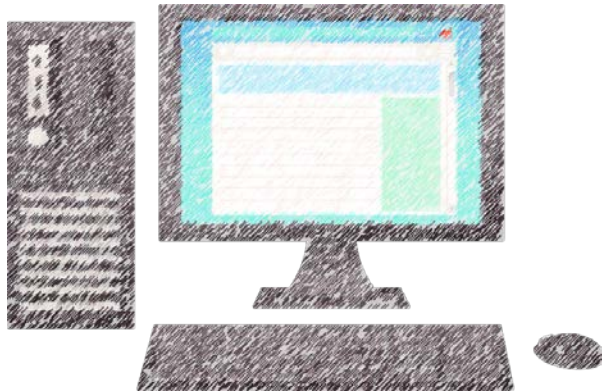


桃園市政府勞動局 112 年勞工學苑產業應用班
Python 程式設計：從零基礎入門到進階

第 3 單元

Python 基礎



林柏江老師

元智大學 電機工程學系 助理教授

pclin@saturn.yzu.edu.tw

預計課程進度

週次	日期	上午課程內容 (09:00 ~ 12:00)	下午課程內容 (13:00 ~ 16:00)
1	2023/07/23	01. 運算思維簡介	02. Python 快速上手
2	2023/07/30	03. Python 基礎	04. Python 基本資料結構
3	2023/08/06	05. 字串	06. 字典
4	2023/08/13	07. 流程控制	08. 函式
5	2023/08/20	09. 模組與作用域	10. Python 程式檔
6	2023/08/27	11. 檔案系統的使用與檔案讀寫	12. 例外處理
7	2023/09/03	13. 類別與物件導向程式設計	14. 初探資料結構與演算法
8	2023/09/10	15. 陣列	16. 鏈結串列
9	2023/09/17	17. 堆疊與佇列	18. 圖形結構
	2023/09/24, 2023/10/01, 2023/10/08 (共三周) 放假		
10	2023/10/15	19. 樹狀結構	20. 分治法
11	2023/10/22	21. 動態規劃	22. 貪婪演算法
12	2023/10/29	23. 回溯	24. 分支定界法

課程大綱

1. 縮排及區塊結構
2. 註解
3. 變數及其設定 assignment
4. 運算式
5. 字串
6. 數字
7. None 值
8. 用 `input()` 取得使用者的輸入
9. 內建算符
10. 基本 Python 風格與命名慣例

課程大綱

1. 縮排及區塊結構
2. 註解
3. 變數及其設定 assignment
4. 運算式
5. 字串
6. 數字
7. None 值
8. 用 `input()` 取得使用者的輸入
9. 內建算符
10. 基本 Python 風格與命名慣例

Python 的縮排及區塊結構

- Python 的程式敘述是以行為單位，一行就是一個敘述。
- Python 沒有分行符號。
- Python 使用空格和縮排來區分程式碼的區塊結構，而不是使用大括號。
- 區塊結構包括：
 - 迴圈的主體
 - 條件式 `if-else` 子句的程式碼
 - ...
- 右圖是計算 9 階乘的 Python 程式碼。
 - 最下面兩行有縮排。
 - Python 直譯器會視它們為 `while` 迴圈的主體。

```
Example3_1_1.py
n = 9
r = 1
while n > 0:
    r = r * n
    n = n - 1
```

Python 的縮排原則

- 可以用任意數量的空格或定位字元 (Tab) 來縮排，只要同一區塊的縮排都一樣即可。
- Python 官方建議使用 4 個空格來縮排。
- VS Code 已內建設定按下一個 Tab 字元會自動轉換為 4 個空格，並提供自動縮排的功能。
- 有些編輯器中，Tab 字元看起來像是 4 個空格。
- 若縮排混用了 Tab 與空格，人眼可能會看不出來，但是程式會出現 IndentationError 或是 TabError 錯誤訊息。

使用縮排來安排程式結構的好處

- 不會有大括號多寫或少寫的情況，也不需要上下翻找程式碼，去尋找匹配的大括號。
- 程式碼視覺上的結構可反映它的真實結構，只需用人眼看就可以輕鬆掌握程式碼的架構。
- 程式碼撰寫風格維持一致。

Tabs vs. Spaces

[Silicon Valley - S03E06: Tabs versus Spaces](#)

課程大綱

1. 縮排及區塊結構
2. 註解
3. 變數及其設定 assignment
4. 運算式
5. 字串
6. 數字
7. None 值
8. 用 `input()` 取得使用者的輸入
9. 內建算符
10. 基本 Python 風格與命名慣例

註解

- Python 程式碼在大多數情況下，# 符號後面的任何內容都是註解。
- 註解會被 Python 直譯器忽略。
- 若是字串中的 #，會被當作是該字串的一個字元。

Example3_2_1.py

```
# 設定 x 變數的值為 5  
x = 5  
x = 3 # 現在 x 變數值等於 3  
x = "# This is not a comment"
```

課程大綱

1. 縮排及區塊結構
2. 註解
3. 變數及其設定 assignment
4. 運算式
5. 字串
6. 數字
7. None 值
8. 用 `input()` 取得使用者的輸入
9. 內建算符
10. 基本 Python 風格與命名慣例

為什麼要使用變數

- 寫程式時若不使用變數，例如左下範例，半徑 10 以及圓周率 3.14 出現在多個位置。如果將來要修改半徑，或是要使用更精確的圓周率時，需要修改程式碼多個位置。
- 使用變數比較方便，如右下範例。

未使用變數

```
print('圓半徑：', 10)
print('圓周長：', 2 * 3.14 * 10)
print('圓面積：', 3.14 * 10 * 10)
```

使用變數

```
radius = 10
PI = 3.14
print('圓半徑：', radius)
print('圓周長：', 2 * PI * radius)
print('圓面積：', PI * radius * radius)
```

Python 的變數

- 在 C 以及其他許多程式語言中，變數能夠儲存的值，只能屬於變數宣告時所指定的資料型別。
- Python 屬於動態定型語言，建立變數不必宣告資料型別，只要命名變數並以 = 符號設定 (指派) 一個值，就建立一個變數。
- 在 Python 中，我們可以指派任何資料型別的物件給變數。

```
>>> x = "Hello"  
>>> print(x)  
Hello  
>>> x = 5  
>>> print(x)  
5
```

變數 **x** 參照 (reference) 到一個字串物件 "Hello"。

變數 **x** 改為參照到另一個整數物件 5。

Python 變數的資料型別是根據它所參照的物件而定。參照到什麼物件，它就是什麼資料型別。

Python 的變數可視為標籤

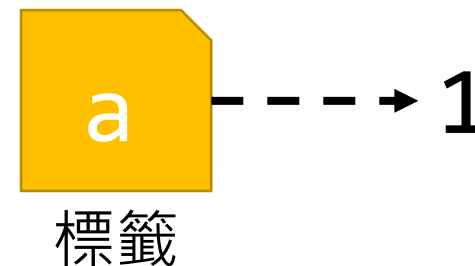
- 在 C 以及其他許多程式語言中，一個變數的運作方式可視為一個水桶，它是可以儲存一個值的容器。

`a = 1` 相當於把整數 1 存放到水桶 a 裡。



- 在 Python 中，變數的運作方式並不像水桶，而像是參照 (reference) 到某個物件的標籤。

`a = 1` 相當於先建立一個整數物件 1，然後把變數 a 參照到這個整數物件 1。

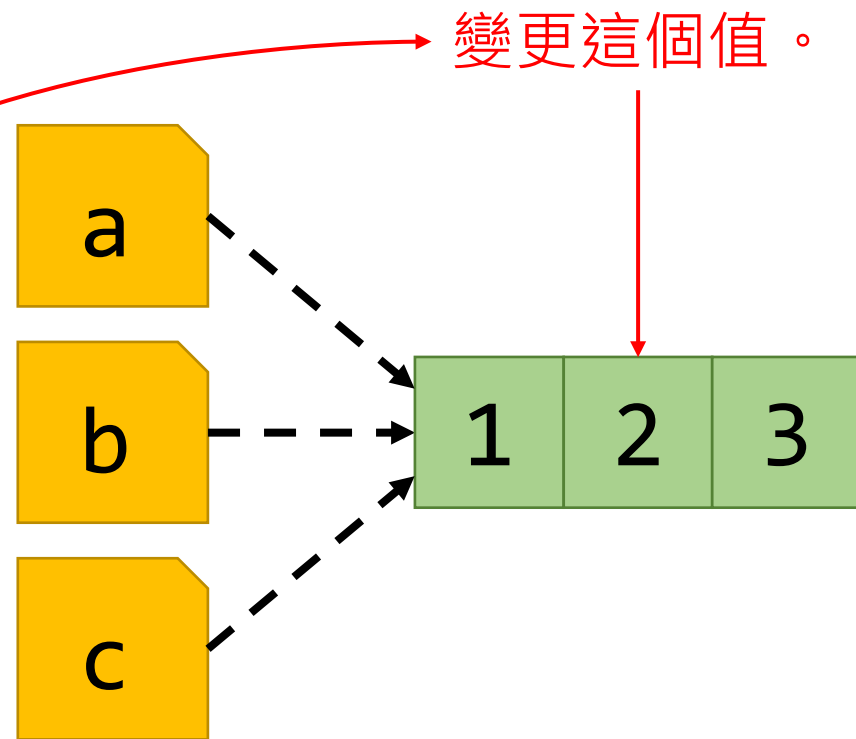


多個變數可以參照到同一個物件

這是 Python 的 list (串列)，類似陣列。
下一章會詳細介紹。

```
>>> a = [1, 2, 3]
>>> b = a
>>> c = a
>>> b[1] = 5
>>> print(a, b, c)
[1, 5, 3] [1, 5, 3] [1, 5, 3]
```

所有變數參照的值也隨之變化。

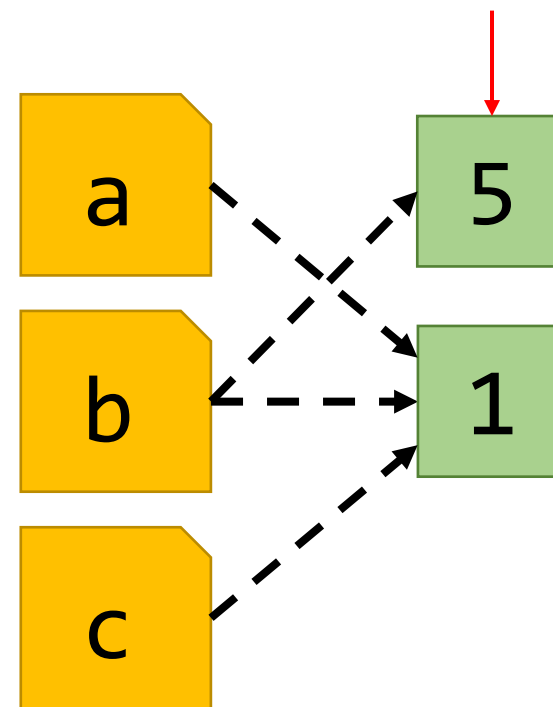


若變數參照的是可變物件 (例如 list)，
修改變數會直接修改這個物件。

多個變數可以參照到同一個物件 (續)

建立新的整數物件 5。

```
>>> a = 1
>>> b = a
>>> c = b
>>> b = 5
>>> print(a, b, c)
1 5 1
```



變數 a、c 參照的值並沒有變化。

若變數參照的是不可變物件 (例如整數 1)，修改變數會建立一個新物件，並把變數改為參照到這個新物件。

Python 變數的命名規則

- 變數名稱有區分大小寫。
 - 例如：`var` 以及 `Var` 是兩個不同的變數。
- 可以使用任何英文字母 (A ~ Z, a ~ z)、數字 (0 ~ 9) 以及底線 (_) 來組成。
- 必須以字母或底線開頭，不能以數字開頭。
- 底線開頭或結尾的名稱，在 Python 中有特殊用途。為變數命名時，請先不要以底線開頭或結尾。

Python 的保留字

以下是 Python 的保留字 (reserved words，也稱作 keywords)，不能作為變數的名稱。

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

如何判斷一個字串 是不是 Python 的保留字？

keyword 模組的 `iskeyword()` 方法可用來判斷一個字串是不是 Python 的保留字。

```
>>> import keyword
>>> keyword.iskeyword('for')
True
>>> keyword.iskeyword('my_variable')
False
```

使用 `del` 敘述刪除變數

- 變數建立後，可使用 `del` 敘述刪除它。
- 刪除後，若嘗試存取這個變數的內容，會導致錯誤。

```
>>> x = 5
>>> print(x)
5
>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>>
```

Python 的例外 (Exception)

- Python 官方說明文件 Built-in Exceptions 中記載了所有錯誤的完整列表，以及發生的原因。

<https://docs.python.org/3/library/exceptions.html>

Python » English » 3.11.4 » 3.11.4 Documentation » The Python Standard Library » Built-in Exceptions | [previous](#) | [next](#) | [modules](#) | [index](#)

Table of Contents

- Built-in Exceptions
 - Exception context
 - Inheriting from built-in exceptions
 - Base classes
 - Concrete exceptions
 - OS exceptions
 - Warnings
 - Exception groups
 - Exception hierarchy

Previous topic

Built-in Types

Next topic

Text Processing Services

This Page

- [Report a Bug](#)
- [Show Source](#)

Built-in Exceptions

In Python, all exceptions must be instances of a class that derives from `BaseException`. In a `try` statement with an `except` clause that mentions a particular class, that clause also handles any exception classes derived from that class (but not exception classes from which *it* is derived). Two exception classes that are not related via subclassing are never equivalent, even if they have the same name.

The built-in exceptions listed below can be generated by the interpreter or built-in functions. Except where mentioned, they have an “associated value” indicating the detailed cause of the error. This may be a string or a tuple of several items of information (e.g., an error code and a string explaining the code). The associated value is usually passed as arguments to the exception class’s constructor.

User code can raise built-in exceptions. This can be used to test an exception handler or to report an error condition “just like” the situation in which the interpreter raises the same exception; but beware that there is nothing to prevent user code from raising an inappropriate error.

The built-in exception classes can be subclassed to define new exceptions; programmers are encouraged to derive new exceptions from the `Exception` class or one of its subclasses, and not from `BaseException`. More information on defining exceptions is available in the Python Tutorial under [User-defined Exceptions](#).

Exception context

課程大綱

1. 縮排及區塊結構
2. 註解
3. 變數及其設定 assignment
4. 運算式
5. 字串
6. 數字
7. None 值
8. 用 `input()` 取得使用者的輸入
9. 內建算符
10. 基本 Python 風格與命名慣例

運算式 (Expression)

- 範例：以下程式碼計算 3 和 5 的平均值，並把結果保留在變數 `z` 中。

```
x = 3
y = 5
z = (x + y) / 2
```

- 從 Python 3 開始，除法 `/` 會傳回一個浮點數。
- 如果希望得到除法結果小於商數的最大整數，可以用 `//` 計算符號。

```
>>> 3 / 2
1.5
>>> 3 // 2
1
>>> -1 // 2
-1
```

運算式的優先順序

- 數學四則運算的優先順序標準規則適用於 Python 運算式。
- 試試看：
 - $x = 2 + 4 * 5 - 6 / 3$ 會得到什麼結果？
 - 在 Python Shell 輸入以上運算式，驗證答案是否正確。

課程大綱

1. 縮排及區塊結構
2. 註解
3. 變數及其設定 assignment
4. 運算式
5. 字串
6. 數字
7. None 值
8. 用 `input()` 取得使用者的輸入
9. 內建算符
10. 基本 Python 風格與命名慣例

字串

- 一對單引號 ' ' 或是雙引號 " " 框起來的文字是一個字串。
- 字串的資料型別是 `str`。

```
>>> "Po-Chiang Lin"  
'Po-Chiang Lin'  
>>> 'pclin'  
'pclin'  
>>> type('pclin')  
<class 'str'>
```

字串中的跳脫序列 (Escape Sequence)

- 以單引號框起來的字串，如果在字串資料中遇到單引號時，則必須以跳脫序列 \ 來跳脫 (escape) 處理，避免被誤認為字串結束。
- 同理，以雙引號框起來的字串，如果在字串資料中遇到雙引號時，則必須以跳脫序列 \ 來跳脫處理，避免被誤認為字串結束。
- 若是字串中有 \ 字元時，通常必須再用一個跳脫序列 \ 來跳脫處理。
- \n 表示換行。
- \t 表示 Tab 字元。

```
>>> print('I'm Po-Chiang Lin')
File "<stdin>", line 1
      print('I'm Po-Chiang Lin')
            ^
SyntaxError: invalid syntax
>>> print('I\'m Po-Chiang Lin')
I'm Po-Chiang Lin
>>> print("I love "Python"!")
File "<stdin>", line 1
      print("I love "Python"!")
            ^
SyntaxError: invalid syntax
>>> print("I love \"Python\"!")
I love "Python"!
>>> print('path = C:\\Users\\pclin')
path = C:\Users\pclin
```

字串

- 雙引號的字串中可以直接帶單引號，不需要使用跳脫序列 \ 來跳脫處理。
- 單引號的字串中可以直接帶雙引號，不需要使用跳脫序列 \ 來跳脫處理。

```
>>> print("I'm Po-Chiang Lin")  
I'm Po-Chiang Lin  
>>> print('I love "Python"!')  
I love "Python"!
```

建立跨行字串：方法 1

- Python 提供了三引號字串，可用來建立跨行字串，而且字串中可以帶單引號以及雙引號，不需要使用跳脫序列 \ 來跳脫處理。
- 此方法的缺點是會把換行字元 \n 也包含進來。

```
>>> long_string = """Lorem ipsum dolor sit amet, consectetur
... adipiscing elit, sed do eiusmod tempor incididunt ut labore
... et dolore magna aliqua."""
>>> long_string
'Lorem ipsum dolor sit amet, consectetur \nadipiscing elit, sed do
eiusmod tempor incididunt ut labore \net dolore magna aliqua.'
```

建立跨行字串：方法 2

- 使用 \ 來定義長字串。
- \ 後面必須立刻換行，不能有空格或任何字元。
- 以下例子的三行字串會視為一行。

```
>>> long_string = "Lorem ipsum dolor sit amet, consectetur " \
...               "adipisicing elit, sed do eiusmod tempor " \
...               "incididunt ut labore et dolore magna aliqua."
>>> long_string
'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua.'
```

建立跨行字串：方法 3

- Python 會把括號內的敘述視為一行。
- 多個空白相隔的字串也會自動連接起來。

```
>>> long_string = (  
...     "Lorem ipsum dolor sit amet, consectetur adipisicing "  
...     "elit, sed do eiusmod tempor incididunt ut labore et "  
...     "dolore magna aliqua.")  
>>> long_string  
'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod  
tempor incididunt ut labore et dolore magna aliqua.'
```

字串與數值

- 可以使用 `str()` 把數值轉換成字串。
- 可以使用 `ord()` 取得某個字元的碼點。
- 可以使用 `chr()` 把碼點轉換成字元。

```
>>> str(3.14)
'3.14'
>>> ord('林')
26519
>>> hex(ord('林'))
'0x6797'
>>> chr(26519)
'林'
```


字串的索引

- 想要取得字串中某個位置的字元時，可以使用索引。
- Python 的索引從 0 開始。
- 想要測試某個字元是否在字串中，可以使用 `in`。

```
>>> text = '哈囉'
>>> text[0]
'哈'
>>> text[1]
'囉'
>>> '哈' in text
True
>>> '林' in text
False
```

字串格式化

字串格式化範例

```
>>> name = 'pclin'
>>> print('Hi', name)
Hi pclin
>>> print('Hi %s' %name)
Hi pclin
>>> print('Hi {}'.format(name))
Hi pclin
>>> print(f'Hi {name}')
Hi pclin
>>>
```

舊式

新式

f-string

字串格式化範例

```
>>> name = 'pclin'
>>> name2 = 'wang'
>>> print('Hi', name, name2)
Hi pclin wang
>>> print('Hi %s %s' %(name,name2))
Hi pclin wang
>>> print('Hi {} {}'.format(name, name2))
Hi pclin wang
>>> print('Hi {1} {0}'.format(name, name2))
Hi wang pclin
>>> print('Hi {0} {0}'.format(name, name2))
Hi pclin pclin
>>> print(f'Hi {name} {name2}')
Hi pclin wang
>>>
```

課程大綱

1. 縮排及區塊結構
2. 註解
3. 變數及其設定 assignment
4. 運算式
5. 字串
6. 數字
7. None 值
8. 用 `input()` 取得使用者的輸入
9. 內建算符
10. 基本 Python 風格與命名慣例

Python 的數值資料型別

- 整數：
 - 例如 0、-11、+33、123456 等。
 - 整數位數無上限，僅受機器資源的限制。
- 浮點數：
 - 可以用小數點或科學記法表示。
 - 例如 3.14、-2e-8、2.718281828 等。
 - 精度由底層機器控制，但通常等於 C 語言的 double (64 位元) 資料型別。
- 複數：
 - 包括實部與虛部。
 - 例如 $3 + 2j$ 。
- 布林值：
 - 只有 True 以及 False 兩種值。
 - 行為與 1 以及 0 相同。

數值型態：整數型態

- 型態為 `int`，不再區分整數與長整數。
- 整數的長度不受限制 (除了硬體上的限制之外)。
- 直接寫一個整數值，預設是十進位 (decimal) 數字。
- 二進位 (binary) 數字要在前面放 `0b` 或是 `0B`。
- 八進位 (octal) 數字要在前面放 `0o` 或是 `0O`。
- 十六進位 (hexadecimal) 數字要在前面放 `0x` 或是 `0X`。

```
>>> 10
10
>>> 0b1010
10
>>> 0o12
10
>>> 0xA
10
```

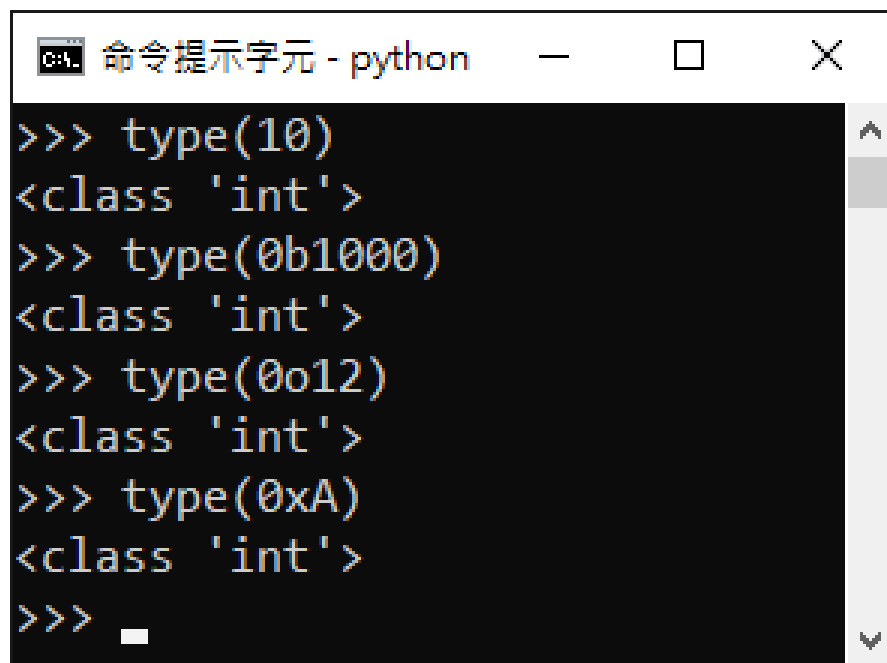
數字太長時的處理方法

從 Python 3.6 開始，若數字太長時可用底線隔開，方便撰寫與閱讀。

```
>>> 1_000_000_000_000_000_000
1000000000000000
>>> 0xFF_FF_FF_FF
4294967295
>>> 0b_1001_0100_1011_0011
38067
```

如何得知某個資料的型態？

- 使用 `type()` 可以取得某個資料的型態。



```
命令提示字元 - python
>>> type(10)
<class 'int'>
>>> type(0b1000)
<class 'int'>
>>> type(0o12)
<class 'int'>
>>> type(0xA)
<class 'int'>
>>> _
```

從其他型態建立整數

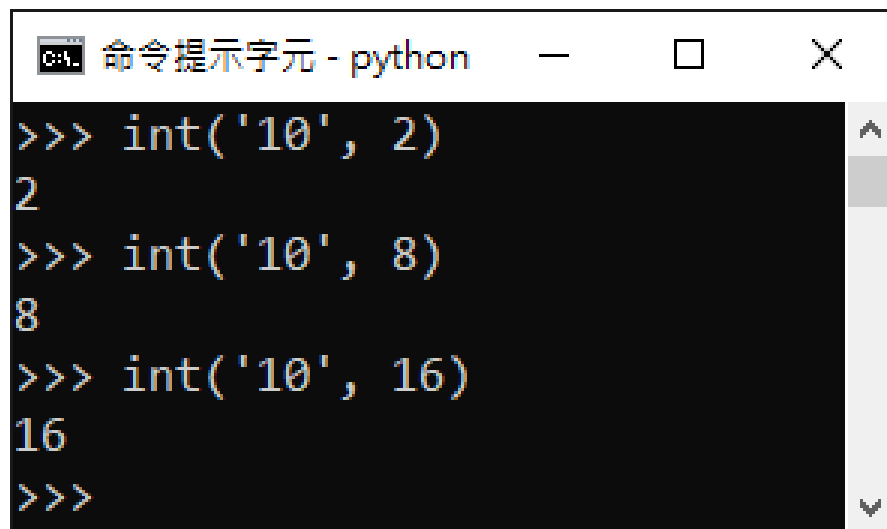
- 使用 `int()` 可以從字串、浮點數、布林等型態建立整數。
- 使用 `bin()`、`oct()`、`hex()` 可以把十進位整數分別以二進位、八進位、十六進位表示的字串傳回。

```
命令提示字元 - python
>>> int(10)
10
>>> int(3.14)
3
>>> int(True)
1
>>> int(False)
0
>>>
```

```
命令提示字元 - python
>>> bin(10)
'0b1010'
>>> oct(10)
'0o12'
>>> hex(10)
'0xa'
>>>
```


指定基底

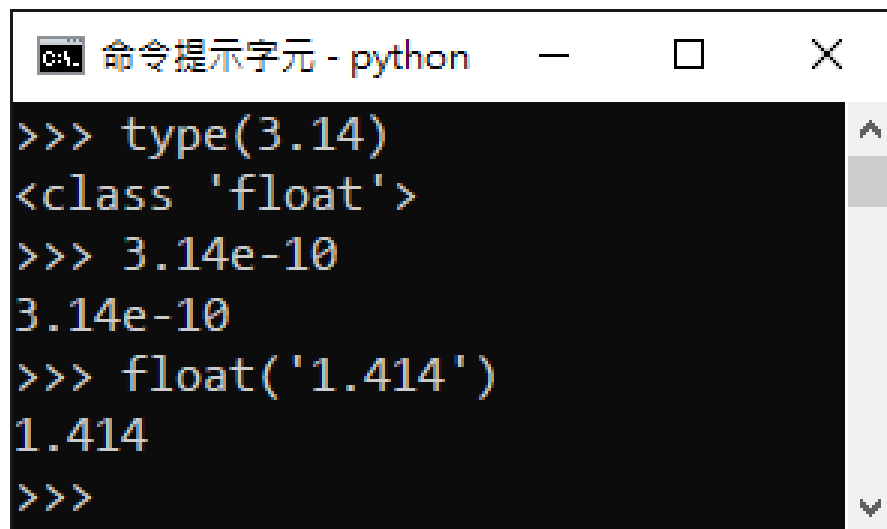
- 使用 `int()` 從字串建立整數時，預設是把字串用十進位來解析。
- 也可以指定基底。

A screenshot of a Python command prompt window titled "命令提示字元 - python". The window has a black background with yellow text. It shows three lines of code being executed: `>>> int('10', 2)` returns `2`, `>>> int('10', 8)` returns `8`, and `>>> int('10', 16)` returns `16`. The prompt `>>>` is shown at the end of the last line.

```
>>> int('10', 2)
2
>>> int('10', 8)
8
>>> int('10', 16)
16
>>>
```

數值型態：浮點數型態

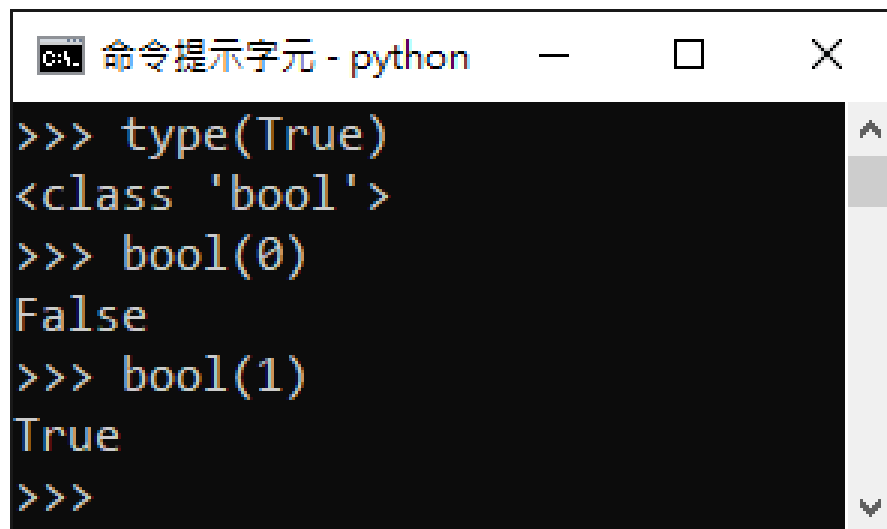
- 浮點數是 float 型態。
- 可以使用 3.14e-10 這樣的表示法。
- 可以使用 float() 把字串解析為浮點數。



```
命令提示字元 - python
>>> type(3.14)
<class 'float'>
>>> 3.14e-10
3.14e-10
>>> float('1.414')
1.414
>>>
```

數值型態：布林型態

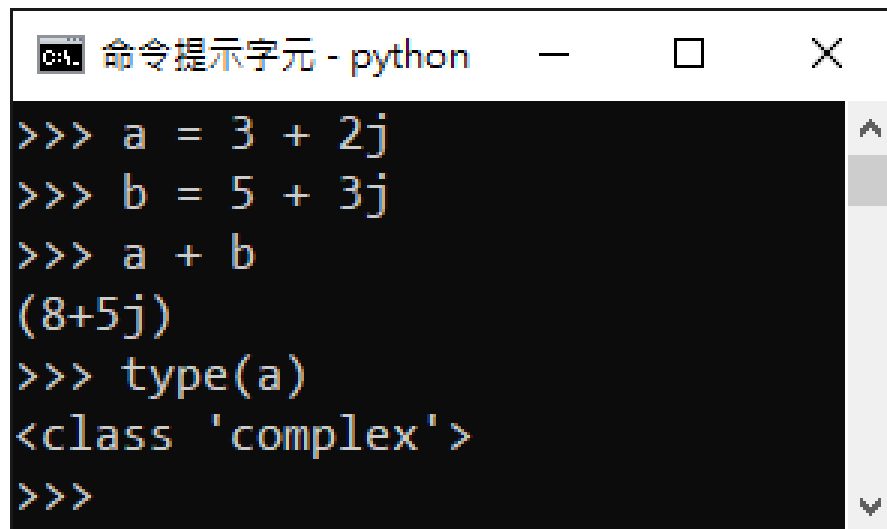
- 布林型態只有兩種值：True 以及 False，為 bool 型態。
- 可以使用 `bool()` 把 0 轉換為 False，把非 0 轉換為 True。

A screenshot of a Windows command prompt window titled "命令提示字元 - python". The window has a black background with yellow text. It shows the following commands and their outputs:

```
>>> type(True)
<class 'bool'>
>>> bool(0)
False
>>> bool(1)
True
>>>
```

複數

- 使用 $a + bj$ 的形式撰寫複數。
- 型態是 `complex`。
- 可直接做算術運算。

A screenshot of a Windows command prompt window titled "命令提示字元 - python". The window has a black background with yellow text. It shows a series of Python commands and their outputs: defining 'a' as 3 + 2j, defining 'b' as 5 + 3j, adding 'a' and 'b' to get (8+5j), and checking the type of 'a' which is <class 'complex'>. The prompt is currently at the start of a new line.

```
>>> a = 3 + 2j
>>> b = 5 + 3j
>>> a + b
(8+5j)
>>> type(a)
<class 'complex'>
>>>
```

數值運算的範例

```
>>> 5 + 2 - 3 * 2
1
>>> 5 / 2 # 除法 ( / ) 運算結果為浮點數
2.5
>>> 5 / 2.0
2.5
>>> 5 // 2 # 使用 // 除法的結果為小於商數的最大整數
2
>>> 5.0 // 2 # 5.0 是浮點數，所以運算結果變成浮點數
2.0
>>> 30000000000 # 很多程式語言中這個數值通常會超出整數型別的範圍
30000000000
>>> 30000000000 * 3
90000000000
>>> 30000000000 * 3.0 # 3.0 是浮點數，所以運算結果是浮點數
90000000000.0
```

數值運算的範例 (續)

```
>>> 2.0e-8 # 用科學計數法表示一個浮點數
2e-08
>>> 3000000 * 3000000
9000000000000
>>> int(200.2)
200
>>> int(2e2)
200
>>> float(200)
200.0
```

課程大綱

1. 縮排及區塊結構
2. 註解
3. 變數及其設定 assignment
4. 運算式
5. 字串
6. 數字
7. **None 值**
8. 用 `input()` 取得使用者的輸入
9. 內建算符
10. 基本 Python 風格與命名慣例

None 值

- None 用於表示空值，類似其他程式語言中的 null 值。
- Python 函式若沒有回傳值，在預設情況下，它會回傳 None。
- 可當作佔位符號 (place holder)，用來標明資料結構中某一個欄位目前尚未有具體的值，亦即先保留這個位置，之後再來填值。
- 整個 Python 系統中只有一個 None 物件。
- 所有對 None 的參照都指向同一個 None 物件。
- 用 == 進行比較時，None 只相等於其本身。

```
>>> None == False
False
>>> None == 0
False
>>> None == None
True
>>> False == 0
True
```


課程大綱

1. 縮排及區塊結構
2. 註解
3. 變數及其設定 assignment
4. 運算式
5. 字串
6. 數字
7. None 值
8. 用 `input()` 取得使用者的輸入
9. 內建算符
10. 基本 Python 風格與命名慣例

用 `input()` 取得使用者的輸入

- `input()` 函式用來取得使用者的輸入。
- 可帶入提示字串參數，用來顯示提示訊息給使用者。

```
>>> name = input("Name? ")
Name? Jane
>>> print(name)
Jane
```

對 `input()` 取得的輸入進行轉換

- `input()` 函式取得的輸入是字串型別。
- 若要把它當作數字來使用，可使用 `int()` 或是 `float()` 等函式進行轉換。

```
>>> age = int(input("Age? "))
Age? 28
>>> print(age+1)
29
```

課程大綱

1. 縮排及區塊結構
2. 註解
3. 變數及其設定 assignment
4. 運算式
5. 字串
6. 數字
7. None 值
8. 用 `input()` 取得使用者的輸入
9. 內建算符
10. 基本 Python 風格與命名慣例

Python 內建運算符號

- 加減乘除運算
- 比較與指定運算
- 邏輯運算
- 位元運算
- 索引切片運算

加減乘除運算

- 使用 $+$ 、 $-$ 、 $*$ 、 $/$ 運算子。
- 好像很簡單！？
- 試試看，以下的運算會得到什麼結果？

```
0.1 + 0.1 + 0.1  
1.0 - 0.8  
0.1 + 0.1 + 0.1 == 0.3
```

加減乘除運算

- 如果需要處理小數而且需要精確的結果，可以使用 `decimal.Decimal` 類別。

範例程式

```
import decimal

n1 = '0.1'
n2 = '0.3'
d1 = decimal.Decimal(n1)
d2 = decimal.Decimal(n2)
n1 = float(n1)
n2 = float(n2)

print('不使用 decimal')
print(f'{n1} + {n1} + {n1} = {n1 + n1 + n1}')
print(f'{n1} + {n1} + {n1} == {n2} is {n1 + n1 + n1 == n2}')

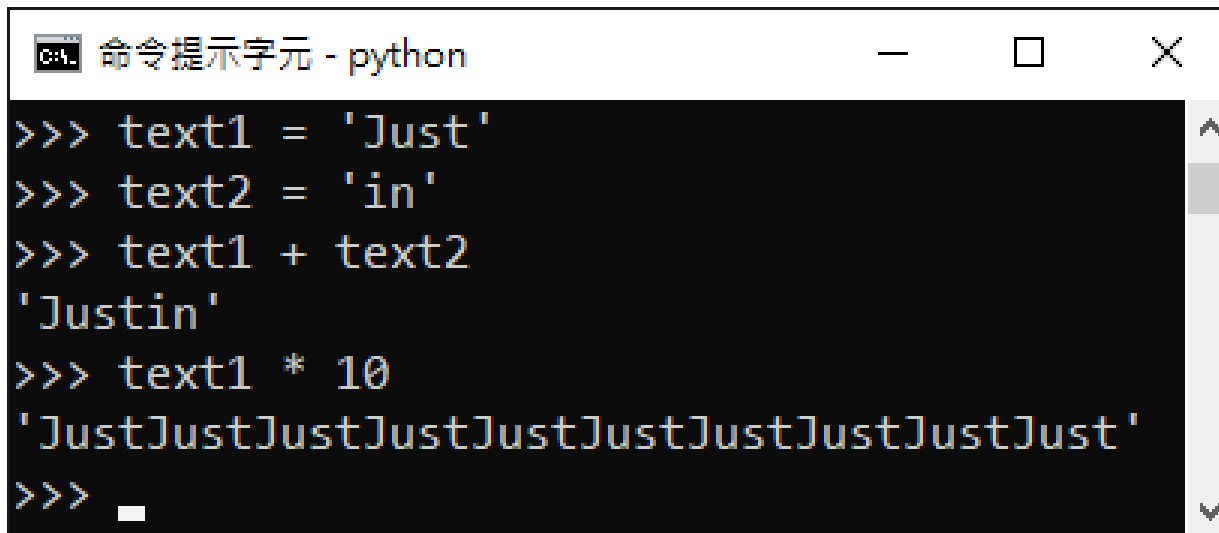
print('使用 decimal')
print(f'{d1} + {d1} + {d1} = {d1 + d1 + d1}')
print(f'{d1} + {d1} + {d1} == {d2} is {d1 + d1 + d1 == d2}')
```

執行結果

```
不使用 decimal
0.1 + 0.1 + 0.1 = 0.30000000000000004
0.1 + 0.1 + 0.1 == 0.3 is False
使用 decimal
0.1 + 0.1 + 0.1 = 0.3
0.1 + 0.1 + 0.1 == 0.3 is True
```

加減乘除運算

- + 與 * 也可以用於字串、清單、以及元組。
- 使用 + 運算子可以串接。
- 使用 * 運算子可以重複。

A screenshot of a Windows command prompt window titled "命令提示字元 - python". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The background is black, and the text is white. The prompt shows a series of Python commands and their outputs: first, 'text1' is assigned the value 'Just', then 'text2' is assigned 'in'. The next command is 'text1 + text2', which outputs 'Justin'. Then, 'text1 * 10' is entered, resulting in the output 'JustJustJustJustJustJustJustJustJustJust'. The prompt ends with three greater-than signs and a cursor.

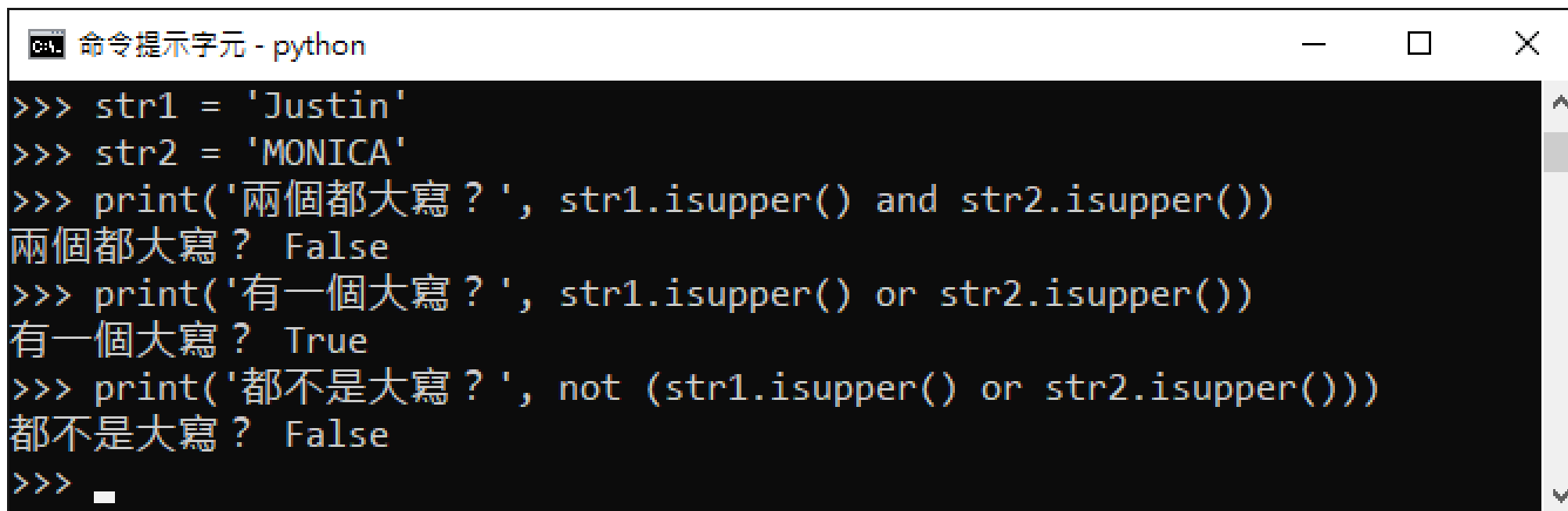
```
>>> text1 = 'Just'
>>> text2 = 'in'
>>> text1 + text2
'Justin'
>>> text1 * 10
'JustJustJustJustJustJustJustJustJustJust'
>>> _
```


比較與指定運算

- $>$ 、 \geq 、 $<$ 、 \leq 、 $==$ 、 $!=$ 、 $<>$
- $<>$ 效果與 $!=$ 相同，不過建議使用 $!=$ 比較清楚，不要再用 $<>$ 。
- 可以串接在一起。
 - $x < y \leq z$ 相當於 $x < y$ and $y \leq z$
 - $w == x == y == z$

邏輯運算

- 且：and
- 或：or
- 反相：not

A screenshot of a Windows command prompt window titled "命令提示字元 - python". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The background is black, and the text is white. The code being executed is as follows:

```
>>> str1 = 'Justin'
>>> str2 = 'MONICA'
>>> print('兩個都大寫?', str1.isupper() and str2.isupper())
兩個都大寫? False
>>> print('有一個大寫?', str1.isupper() or str2.isupper())
有一個大寫? True
>>> print('都不是大寫?', not (str1.isupper() or str2.isupper()))
都不是大寫? False
>>> _
```

位元運算

- AND: &
- OR: |
- XOR: ^
- NOT: ~

範例程式

```
print('AND運算 : ')
print('0 AND 0 {:5d}'.format(0 & 0))
print('0 AND 1 {:5d}'.format(0 & 1))
print('1 AND 0 {:5d}'.format(1 & 0))
print('1 AND 1 {:5d}'.format(1 & 1))

print('\nOR運算 : ')
print('0 OR 0 {:6d}'.format(0 | 0))
print('0 OR 1 {:6d}'.format(0 | 1))
print('1 OR 0 {:6d}'.format(1 | 0))
print('1 OR 1 {:6d}'.format(1 | 1))

print('\nXOR運算 : ')
print('0 XOR 0 {:5d}'.format(0 ^ 0))
print('0 XOR 1 {:5d}'.format(0 ^ 1))
print('1 XOR 0 {:5d}'.format(1 ^ 0))
print('1 XOR 1 {:5d}'.format(1 ^ 1))
```

執行結果

```
AND運算 :
0 AND 0      0
0 AND 1      0
1 AND 0      0
1 AND 1      1

OR運算 :
0 OR 0      0
0 OR 1      1
1 OR 0      1
1 OR 1      1

XOR運算 :
0 XOR 0      0
0 XOR 1      1
1 XOR 0      1
1 XOR 1      0
```

位元運算

- 左移： \ll
- 右移： \gg

範例程式

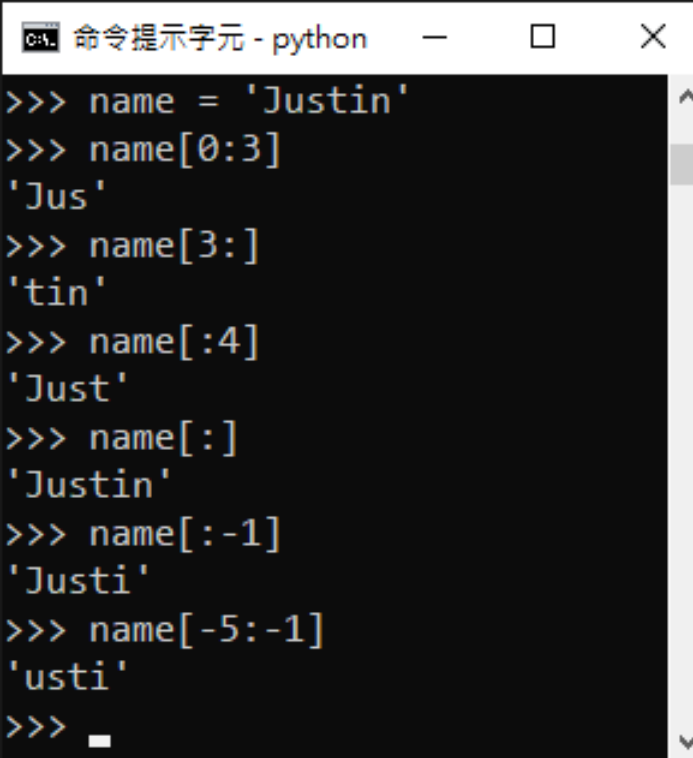
```
number = 1
print('2 的 0 次方: ', number)
print('2 的 1 次方: ', number << 1)
print('2 的 2 次方: ', number << 2)
print('2 的 3 次方: ', number << 3)
```

執行結果

```
2 的 0 次方: 1
2 的 1 次方: 2
2 的 2 次方: 4
2 的 3 次方: 8
```

索引切片運算

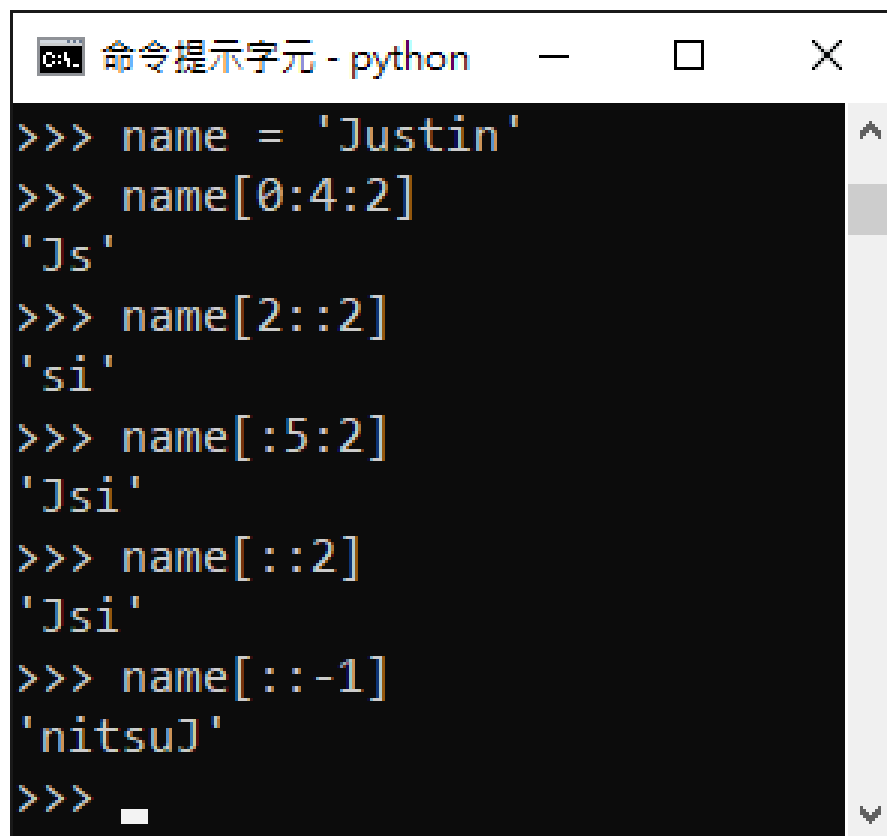
- 在 Python 的內建型態中，只要具有索引特性，基本上都能進行切片運算，包括字串 (string)、串列 (list，也稱作清單)、元組 (tuple) 等等。
- 可使用 [start:end] 形式。
 - start 是開始索引。
 - end 是結尾索引。
- 切片結果包括起始索引，但不包括結尾索引。



```
命令提示字元 - python
>>> name = 'Justin'
>>> name[0:3]
'Jus'
>>> name[3:]
'tin'
>>> name[:4]
'Just'
>>> name[:]
'Justin'
>>> name[:-1]
'Justi'
>>> name[-5:-1]
'usti'
>>> 
```

索引切片運算

- 也可以使用 [start:end:step] 形式：



```
命令提示字元 - python
>>> name = 'Justin'
>>> name[0:4:2]
'Js'
>>> name[2::2]
'si'
>>> name[:5:2]
'Jsi'
>>> name[::2]
'Jsi'
>>> name[::-1]
'nitsuJ'
>>> _
```

課程大綱

1. 縮排及區塊結構
2. 註解
3. 變數及其設定 assignment
4. 運算式
5. 字串
6. 數字
7. None 值
8. 用 `input()` 取得使用者的輸入
9. 內建算符
10. 基本 Python 風格與命名慣例

基本 Python 風格與命名慣例

- Python 對程式碼寫作風格的限制相對較少。
- 但 Python 要求使用縮排，把程式碼組織成區塊。
 - 沒有強制一定要用哪種縮排字元，使用空格或 Tab 都可以。
 - 也沒有強制縮排字元的數量。
- Python 有一些建議的程式撰寫風格慣例，記錄於 PEP 8 (Python Enhancement Proposal 8): Style Guide for Python Code 中。
<https://www.python.org/dev/peps/pep-0008/>
- 教科書的附錄 A 節錄了 PEP 8 Python 程式碼撰寫風格。

Guido van Rossum 的一項重要見解

Code is read much more often than it is written.

程式碼被閱讀的頻率遠高於它被撰寫的頻率。

PEP 8 旨在提高程式碼的可讀性，使得不同人寫的 Python 程式碼都能保持一致性的風格，以便於閱讀與維護。



Python 的命名慣例

情況	建議	範例
模組/套件名稱	短名詞，全部小寫，必要時才使用底線 (_)	<code>imp, sys</code>
函式名稱	全部小寫，為增加可讀性可使用底線	<code>foo(), my_func()</code>
變數名稱	全部小寫，為增加可讀性可使用底線	<code>my_var</code>
類別名稱	每一個英文單字的第一個字母大寫	<code>MyClass</code>
常數名稱	全部大寫，單字與單字之間加底線	<code>PI, TAX_RATE</code>
縮排	每個級別為 4 個空格，不使用 Tab	
比較	無須用 <code>==</code> 寫出比較結果是 <code>True</code> 還是 <code>False</code>	<pre>if my_var: if not my_var:</pre>

底線開頭或結尾的名稱有特殊用途

名稱	特殊用途
前單底線 <code>_var</code>	模組的私有變數或函式，用 <code>*</code> 號匯入模組時無法匯入以 <code>_</code> 底線開頭的名稱。 類別或物件的私有變數或方法，這是約定成俗的慣例用法，類別或物件外部仍然可以存取。
前雙底線 <code>__var</code>	類別或物件的私有變數或方法，會被 Python 改名，所以物件或類別外部無法直接用原本名稱存取。
前後雙底線 <code>__var__</code>	保留給 Python 內部使用的名稱。 您自己的變數或函式名稱請避免使用這種形式。
後單底線 <code>var_</code>	用來避免與 Python 關鍵字衝突。 例如您很想要用 <code>print</code> 這個名稱，但是 <code>print()</code> 是 Python 的內建函式，為了避免發生衝突，可以取名為 <code>print_</code> 。
<code>_</code>	暫時性或不重要的資料，類似免洗變數用完就丟。

重點整理

- Python 使用縮排來區分區塊結構，官方建議使用 4 個空格縮排。
- Python 的變數是標籤的概念，變數是參照到物件的名稱。
- 字串與數字都是 Python 的內建資料型別，其中數字型別有 4 種類型：整數、浮點數、複數和布林值。
- Python 還有一個特殊的基本資料型別：None，用於表示空值。
- 您可以用 `input()` 函式取得使用者的輸入。
- 底線開頭或結尾的名稱，在 Python 中有特殊用途。在您為變數或函式命名時，請避開這些特殊名稱。

程式練習

請設計一個程式，讓使用者輸入姓名、年齡，計算此人還有幾年就可以退休。

程式執行範例如下：

執行結果

```
請輸入姓名：林柏江  
請輸入年齡：49  
林柏江 還有 11 年就能退休！
```