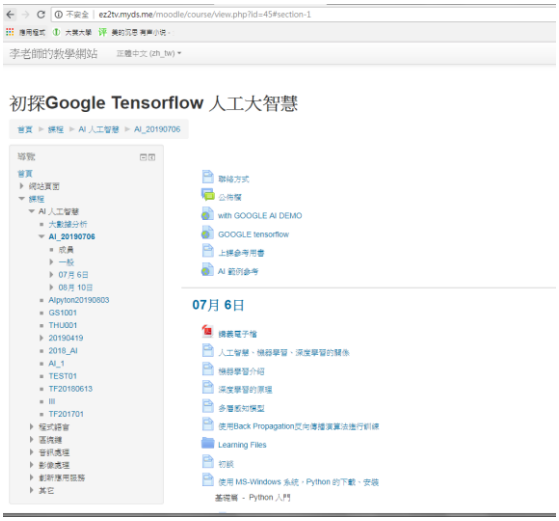
Chapter 2　Python 語法及用法

參考網頁:

- https://blog.kdchang.cc/2016/10/30/python101-tutorial/

- http://ez2tv.myds.me/moodle/course/view.php?id=45#section-1



2.1 基本輸入輸出

在程式執行的過程中，可以使用 input()函式取得使用者的輸入，input()可以指定提示文字，使用者輸入的文字則以字串傳回（Python 2.7 的輸入是使用 `raw_input()`）。例如：

- hello.py

```
name = input('請輸入你的名稱：')
print('歡迎 ', name)
```

預設.py 檔案必須是 UTF-8 編碼。如下執行指令載入指令稿直譯並執行：

```
> python hello.py
請輸入你的名稱：良葛格
歡迎　良葛格
```

如果.py 檔案想要是 UTF-8 以外的編碼，必須在第一行放置編碼聲明（encoding declaration）。例如若為 Big5 編碼的.py 檔案：

- hello.py

```
# coding=Big5
name = input('請輸入你的名稱：')
print('歡迎 ', name)
```

到目前為止，輸出都是使用 print()函式，使用 help(print)查詢其說明：

```
print(value, ..., sep=' ', end='\n', file=sys.stdout)
```

可以看到，除了指定值輸出之外，還可以使用 sep 指定每個輸出值之間的分隔字元，預設值為一個空白，可以使用 end 指定輸出後最後一個字元，預設值是'\n'。例如：

```
>>> print(1, 2, 3)
1 2 3
>>> print(1, 2, 3, sep='')
123
>>> print(1, 2, 3, sep='', end='\n\n')
123
```

```
>>>
```

預設的輸出是系統標準輸出，可以使用 file 指定至其它的輸出。例如以下會將指定的值輸出至 data.txt：

```
>>> print(1, 2, 3, file = open('data.txt', 'w'))
>>>
```

open()函式會開啟檔案，並傳回一個_io.TextIOWrapper 物件，上例將之指定給 print()作為輸出目標。

你可以格式化字串，例如：

```
>>> text = '%d %.2f %s' % (1, 99.3, 'Justin')
>>> print(text)
1 99.30 Justin
>>> print('%d %.2f %s' % (1, 99.3, 'Justin'))
1 99.30 Justin
>>>
```

格式化字串時，所使用的%d、%f、%s 等與 C 語言類似，之後使用%接上一個 tuple，也就是範例中以()包括的實字表示部份。如果你要將使用者的輸入字串轉換為整數、浮點數、布林值等型態，可以使用 int、float、bool 等類別來建構對應物件。例如：

```
>>> int('1')
1
>>> float('3.14')
3.14
>>> bool('true')
True
>>>
```

如果你要將資料寫入檔案或從檔案讀出，可以使用 open()函式：

```
open(file,mode="r",buffering=None,encoding=None,
     errors=None,newline=None,closefd=True)
```

例如，若要讀取檔案：

- show.py

```
name = input('請輸入檔名：')
file = open(name, 'r', encoding='UTF-8')
content = file.read()
print(content)
file.close()
```

read()方法會一次讀取所有的檔案內容，在不使用檔案時，可以使用 close()將檔案關閉以節省資源。如果要逐行讀取，則可以使用 readline()方法。例如：

- show.py

```
name = input('請輸入檔名：')
file = open(name, 'r', encoding='UTF-8')
while True:
    line = file.readline()
    if not line: break
    print(line, end='')
file.close()
```

如果資料讀取完畢，readline()會傳回空字串，這在布林判斷式中會是 false。另一個比較簡潔的方式是使用 for 迴圈。例如：

- show.py

```
name = input('請輸入檔名：')
file = open(name, 'r', encoding='UTF-8')
for line in file.readlines():
    print(line, end='')
file.close()
```

readlines()方法會用一個字串陣列收集讀取的每一行，for 迴圈每次取出字串陣列中的一個字串元素，並使用 print()函式顯示。事實上，更有效率的方式，是呼叫檔案物件的 next()方法，next()方法每次傳回下一行，並在沒有資料可讀取時丟出 StopIteration。可以使用 for 迴圈自動呼叫 next()方法，並在捕捉到 StopIteration 時離開迴圈。例如：

- show.py

```
name = input('請輸入檔名：')
for line in open(name, 'r', encoding='UTF-8'):
    print(line, end='')
```

這樣的寫法會自動關閉檔案，不過牽涉到更多技術細節，現階段你只要記得有這種用法，技術細節在之後的文件還會介紹。

如果要寫資料至檔案，則在使用 open()函式時，指定模式為'w'，並使用 write()方法進行資料寫入。例如：

```
name = input('請輸入檔名：')
file = open(name, 'w', encoding = 'UTF-8')
file.write('test')
file.close()
```

若需要了解更多 open()函式的細節，記得使用 help(open)進行查詢

作業 **2.1**:

(1) 請參考前例，例如使用 print(1, 2, 3, file = open('data.txt', 'w'))在 google colab 建立檔案含有 1 2 3 之資料檔，請問如何使用相同的指令 print，使得資料量由

1 2 3 ➔ 1 2 3
4 5 6

提示: 'w'改為'a'

(2) 承上，使用相同的指令 print，印出原有資料量為

1, 2, 3
4, 5, 6

2.2 字串型態

在 Python 中字串有多種的表示方式。最基本的實字表示方式，就是使用雙引號或單引號來包括字元序列。例如：

```
>>> "Justin"
'Justin'
>>> 'Justin'
'Justin'
>>> "Just'in"
"Just'in"
>>> 'Just"in'
'Just"in'
>>> 'Just' 'in'
'Justin'
```

單引號或雙引號的字串表示，在 Python 中可以交替使用，就像上例中，若要在字串中包括單引號，則使用雙引號包括字元序列，反之亦然。如果有兩個連續的字串實字，Python 會自動將之結合為一個字串。

在 Python 撰寫字串致力於簡潔易讀，字串中若包括\，則會自動轉換為\\，所以可以直接如下撰寫字串：

```
>>> 'c:\workspace'
'c:\\workspace'
>>> 'c:\\workspace'
'c:\\workspace'
>>>
```

在 Python 中，直接撰寫'c:\workspace'，就如同你自行撰寫'c:\\workspace'，前者在撰寫與可讀性上都比較方便。\\是跳離（Escape）字串表示，另外還有一些常用的跳離（Escape）字串表示：

\\	反斜線
\'	單引號 '
\"	雙引號 "
\ooo	以 8 進位數指定字元，最多三位數
\xhh	以 16 進 位數指定字元，至少兩位數
\uhhhh	以 Unicode 16 位元編碼指定字元

\Uhhhh	以 Unicode 32 位元編碼指定字元
\0	空字元
\n	換行
\r	歸位
\t	Tab

如果你想要表示跳離字串表示，例如要表示\t，則必須撰寫'\\t'來表示，這有些不方便，這時你可以使用原始字串（Raw String）表示，只要在字串前加上 r 即可。例如：

```
>>> print('\t')

>>> print('\\t')
\t
>>> print(r'\t')
\t
>>> print('c:\\workspace')
c:\workspace
>>> print(r'c:\\workspace')
c:\\workspace
>>> '\t'
'\t'
>>> r'\t'
'\\t'
>>> 'c:\\workspace'
'c:\\workspace'
>>> r'c:\\workspace'
'c:\\\\workspace'
>>>
```

如果你的字串內容必須跨越數行，則可以使用三重引號。例如：

```
>>> '''Justin is caterpillar!
...     caterpillar is Justin!'''
'Justin is caterpillar!\n    caterpillar is Justin!'
>>> text = '''Justin is caterpillar!
...             caterpillar is Justin!'''
>>> print(text)
Justin is caterpillar!
            caterpillar is Justin!
```

```
>>>
```

你在三重引號之間輸入任何內容，則最後的字串就照單全收，包括換行、縮排等。

如果你想知道字串的長度，則可以使用 len()函式。例如：

```
>>> text = 'Justin'
>>> len(text)
6
>>>
```

你可以使用 for 迴圈逐一取出字串中的字元：

```
>>> for c in 'Justin':
...     print(c, end='-')
...
J-u-s-t-i-n->>>
```

或者是使用 in 運算子測試某個字串是否在原字串中：

```
>>> 'Just' in 'Justin'
True
>>>
```

可以使用+運算子來串接字串，使用*可以重複字串：

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

在 Python 中，字串是不可變動的（Immutable），所以+實際會產生新的字串，在強弱型別的光譜中，Python 比較偏強型別，型態較不能自行轉換，例如 Python 中，不能混合字串與數字進行+運算，你得自己將數字轉為字串，才可以進行字串串接：

```
>>> 'score: ' + 90
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

```
TypeError: Can't convert 'int' object to str implicitly
>>> 'score: ' + str(90)
'score: 90'
>>>
```

如上例所示，你可以使用 str()類別將數值轉換為字串。如果你想知道某個字元的編碼，則可以使用 ord()函式，使用 chr()則可以將指定編碼轉換為字元：

```
>>> ord('元')
20803
>>> chr(20803)
'元'
>>>
```

在 Python3 中，每個字串都是 Unicode，不使用內部編碼表現，而使用 str 實例作為代表。如果想將字串轉為指定的編碼實作，可以使用 encode()方法取得一個 bytes 實例，如果有個 bytes 實例，也可以使用 decode()方法指定編碼取得 str 實例：

```
>>> '元'.encode('big5')
b'\xa4\xb8'
>>> '元'.encode('utf-8')
b'\xe5\x85\x83'
>>> '元'.encode('big5').decode('big5')
'元'
>>>
```

字串是由字元序列所組成(可參考給自學者的 **Python 教學(7)：字串**)，如果你想要取得字串中某個字元，則可以使用[]指定索引，索引從 0 開始。例如：

```
>>> name = 'Justin'
>>> name[0]
'J'
>>> name[1]
'u'
>>> name[-1]
'n'
>>>
```

Python 中的索引，不僅可指定正值，還可以指定負值，實際上了解索引意義的開發人員，都知道索引其實就是相對第一個元素的偏移值，在 Python 中，正值索引就是指正偏移值，負值索引就是負偏移值，也就是-1 索引就是倒數第一個元素，-2 索引就是倒數第二個元素。

[]運算子還可以進行切片（Slice）運算。例如：

```
>>> name[0:3]
'Jus'
>>> name[3:]
'tin'
>>> name[:4]
'Just'
>>> name[:-1]
'Justi'
>>> name[-5:-1]
'usti'
>>>
```

上例中指出了切片運算，可以指定起始索引（包括）與結尾索引（不包括）來切出子字串。如果只指定起始索引，不指定結尾索引，則表示切出從起始索引至字串結束間的子字串。如果只指定結尾索引，不指定起始索引，則表示切出從 0 索引至（不包括）結尾索引間的子字串。起始索引與結尾索引都可以指定負值。（[:]則是作淺層複製，只不過對字串這種不可變動的物件沒有實質意義）

切片運算的另一個形式是[i:j:k]，意思是切出起始索引 i 與結尾索引 j（不包括）之間，每隔 k 元素的內容。例如：

```
>>> name[0:4:2]
'Js'
>>> name[2::2]
'si'
>>> name[:5:2]
'Jsi'
>>> name[::2]
'Jsi'
>>> name[::-1]
'nitsuJ'
>>>
```

注意最後一個範例，當間隔指定為正時，表示正偏移每 k 個取出元素，間隔指定為負時，表示負偏移每 k 個取出元素。[::-1]表示從索引 0 至結尾，以負偏移 1 方式取得字串，結果就是反轉字串。

在 Python 中，非物件專屬的操作，是以函式的方式提供，例如之前的 len()函式並不只能用在字串，而可以用在所有的串列物件（只要該物件上有__len__()方法）。物件專屬的操作，則是物件上的方法，例如以下示範幾個字串專屬方法：

```
>>> name.index('i')
4
>>> name.upper()
'JUSTIN'
>>> name.lower()
'justin'
>>> name.startswith('J')
True
>>>
```

相關**字串物件**說明，請參閱網頁 http://yltang.net/tutorial/python/10/

2.3 變數命名規則與縮排

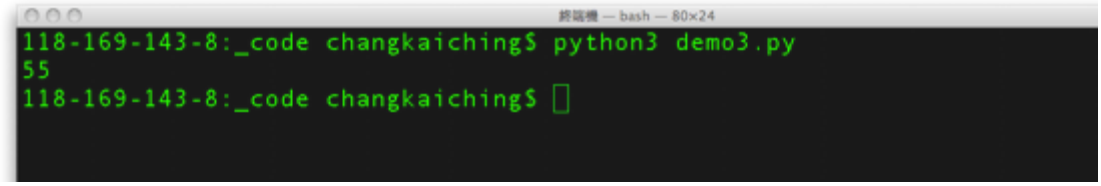
Python 3.x 直接支援 Unicode 編碼，因此除了用為運算子 (operator) 的半形字元或特殊字元外，其他的字元都可以拿來當變數 (variable) 名稱

例如下面使用中文「變數」當變數名稱

?

```
1 變數 = 55
2 print(變數)
```

執行結果如下



雖說中文當變數名稱是可行的，但我們不建議用英文字母、數字與底線以外的字元當變數或識別字 (identifier) 名稱，原因很簡單，因為 Python 社群發展已久的大量程式庫 (library) 中，無論是標準模組庫 (standard library) 或第三方模組庫 (third-party library) 幾乎都是以英文為變數取名，若是要把自己開發的程式與 Python 社群接軌，仍是依社群的習慣較理想。習慣上命名識別字的字元如下表

_

a	b	c	d	e	f	g	h	i	j	k	l	m
n	o	p	q	r	s	t	u	v	w	x	y	z
A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9			

通常是用英文單字或縮寫替識別字命名，注意不能用數字當開頭，以下為合法

的變數名稱

`variable_name`

`_code`

`CODE`

`_1234`

`a1234`

`someThing`

`SomeThing`

函數命名習慣與變數雷同，至於類別 (class) 較常採大寫駝峰型 (upper camel case) ，或用底線連接每個英文單字，例如

`Class_Name`

`ClassName`

類別中的方法 (mehod) 與屬性 (attribute) 則較常採用小寫駝峰型 (lower camel case)，或用底線連接每個英文單字，例如

`method_name`

`methodName`

大體上都是採取有意義的相關英文單字，主要的目的是讓語意清楚。另外須留意 Python 程式裡不能隨意縮排 (indentation) ，因為縮排是 Python 劃分程式區塊 (block) 的方式，例如以下程式

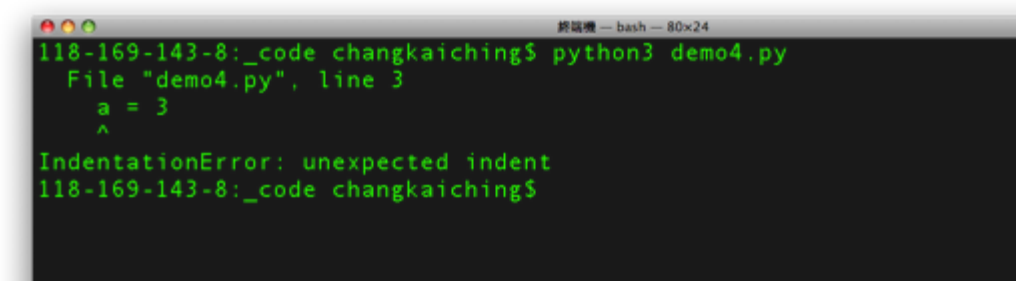
```
a = 1
```

```
print(a)
```

```
 a = 3
```

```
print(a)
```

執行時會發生錯誤



縮排通常會用在控制結構 (control structure) 或函數、類別的定義裡。

2.4 資料型別

在 Python 有以下幾種內建的資料型別，基本資料型別有 Number、String、Boolean

- 數字（Number）

```
num1 = 3
num2 = 2
num3 = num1 / num2
```

- 字串（String）
  字串使用上使用單引號或雙引號成對包起（', "）

```
str = 'data engineer'
# 字串長度
len(str)
# 原始字元
es_str = r'\t'
# 2
len(es_str)
```

- 若是多行的情形：

```
multi_line_str = """
#多行
text = '''Justin is caterpillar!
...          caterpillar is Justin!'''
>>> print(text)
Justin is caterpillar!
       caterpillar is Justin!
```

- 布林值（Boolean）
  決定邏輯判斷，`True` 或 `False`。注意在 Python 中布林值首字是大寫

```
is_num_bigger_than_one = 1 < 2
```

- 在 Python 中 `None` 地位類似於 `null`

```
x = None
print(x == None)
```

- 以下為 Python 的 Falsy 值：
  - False
  - None
  - [] →空的 list
  - {} →空的 dict
  - “” →空的 str
  - set() →空的 set
  - 0
  - 0.0
  - b’’ →空的 byte

  搭配 and, or, not 使用

例如，要查看列表是否為空，而不是像這樣檢查：

```
if len(my_list) != 0:
   print "Not empty!"
```

應這樣做：

```
if my_list:
   print "Not empty!"
```

- 列表（List）
  列表可以說是 Python 中最基礎的一種資料結構。所謂列表指的就是一群按照順序排序的元素（類似於其他程式語言的 array，但多一些額外功能）。

```
list_num = [1, 2, 3]
list = ['string', 1, [], list_num]
list_length = len(list_num)
num_sum = sum(list_num)

print(list_length)
print(num_sum)
```

- 運用 [] 取值（index 從 0 開始）：

```
x = range(10) # [0, 1, 2, ..., 9]
zero = x[0] # 0
nine = x[-1] # 9
x[0] = -1
```

- 切割（[起始 index, 結束 index 但不包含]）：

```
print(x[:3]) # [0, 1, 2] = range(0,3)
print(x[3:]) # [3, 4, 5,..., 9]
print(x[1:5]) # [1, 2, 3, 4]
print(x[0:-1]) # [1, 2, ..., 8]
```

- 檢查元素是否在列表中（逐一檢查，效率較差）：

```
1 in [1, 2, 3] # True
```

- 串接列表：

```
x = [1, 2, 3]
x.extend([4, 5, 6])
```

```
x = [1, 2, 3]
y = x + [4, 5, 6]
```

```
x = [1, 2, 3]
x.append(0) # [1, 2, 3, 0]
```

- 賦值方式：

```
x, y = [1, 2]
```

```
_, y = [1, 2]
```

- 元組（Tuple）
  Tuple 類似於 List 的兄弟，比較大差別在於 Tuple 是 immutable，也就是說宣告後不能修改。列表使用 []，而元組使用 ()

```
my_list = [1, 2]
my_tuple = (1, 2)
my_list[1] = 3

try:
	my_tuple[1] = 4
except TypeError:
	print('cannot modify a tuple')
```

- 多重賦值

```
x, y = 1, 2
x, y = y, x # x == 2, y == 1
```

- 字典（Dictionary）

  或參考 http://yltang.net/tutorial/python/12/

  空字典

  ```
  data={}
  print(data)
  ```

  >>>{}
- 字典裡面添加鍵 keys 值 values，A 代表 keys 而"123"代表 values

  ```
  data={"A":"123"}
  print(data)
  ```

  >>>{'A': '123'}
- 觀看字典裡面的所有 keys 和 values

  ```
  data={"A":"123","B":"456"}
  print(data.keys())
  ```

  >>>dict_keys(['A', 'B'])

  ```
  print(data.values())
  ```

  >>>dict_values(['123', '456'])
- 查詢字典，查詢字典裡面的 key 最後結果是 value 值

  ```
  data={"A":"123","B":"456"}
  print(data['A'])
  ```

  >>>123
- 新增字典的 key 和 value 值使用.setdefault()

```
data={"A":"123","B":"456"}
data.setdefault('C','789')
print(data)
```
>>>{'A': '123', 'B': '456', 'C': '789'}

- 新增字典的 key 和 value 值使用.updata()

```
data={"A":"123","B":"456"}
dic2={"C":"987"}
data.update(dic2)
print(data)
```
>>>{'A': '123', 'B': '456', 'C': '987'}

- 刪除字典的 key 和 value 值使用 del

```
data={"A":"123","B":"456"}
del data['A']
print(data)
```
>>>{'B': '456'}

- 清除字典內的鍵 keys 值 values

```
data.clear()
print(data)
```
>>>{}

- 修改字典的 key 和 value 值

```
data={"A":"123","B":"456"}
data['A']="666"
print(data)
```
>>>{'A': '666', 'B': '456'}

- 複製字典的內容到另一字典

```
data={"A":"123","B":"456"}
data2=data.copy()
print(data2)
```
>>>{'A': '123', 'B': '456'}

- 集合（Set）

  集合類似數學中的集合，裡面包含不重複的元素值

```
s = set()
s.add(1) # { 1 }
s.add(2) # { 1, 2 }
s.add(2) # { 1, 2 }
len(s) # 2
1 in s # True
```

- 集合在判斷元素是否存在的效率相對較好，此外，對於判斷不重複值也很方便

```
list_item = [1, 2, 3, 1, 2, 3]
set_item = set(list_item) # {1, 2, 3}
list(set_item) # [1, 2, 3]
```

- 解析式列表（comprehensive list）

在 Python 我們通常會需要把某個 list 轉換成另外一個 list，比如只挑選其中幾個元素，或是對期中某些元素進行轉換。

```
even_numbers = [x for x in range(5) if x % 2 == 0]
squares = [x for x in range(5) if x % 2 == 0]
even_squares = [x * x for x even_numbers]

# 不操作值的話
zeros = [0 for _ in even_numbers] # 和 even_numbers 長度一樣值都為 0 的串列
```

建立 set 和 dict

```
square_dict = { x : x * x for x in range(5) }
square_set = { x * x for x in [1, -1] } # { 1 }

pairs = [(x, y) for x in range(10) for y in range(10)] # (0, 0), (0, 1)
p = [(x, y) for x in range(3) for y in range( x + 1, 10)]
```

2.4 運算式與陳述

運算式 (expression) 由運算元 (operand) 與運算子 (operator) 所組成

單一運算元就構成一個運算式，運算元可以是變數 (variable) 、字面常數 (literal) 或呼叫函數 (function) ，例如

```
a          # 以變數當運算元
```

```
3     # 以字面常數當運算元
```

```
print() # 呼叫函數
```

運算式，顧名思義，其為運算出一個結果的式子，因此單一運算元的運算式所得到的結果就是運算元的值，如果是函數呼叫，就是函數的回傳值 (return value) 囉！

單一運算元也可以結合單元運算子，單元運算子有

運算子	功能	範例
+	正	+a
-	負	-a

正負如同一般數學表示正負值一樣，同樣的，負負會得正

```
a = -3
```

```
b = +a # b 會等於 -3
```

```
c = -a # c 會等於 3
```

一般運算子都須結合兩個運算元，像是關鍵字 (keyword) 中有邏輯運算子 (logical operator)

運算子	功能	範例
and	且	a and b
or	或	a or b
not	非	not a

not 也是單元運算子， **and** 與 **or** 則需要兩個運算元

```
a = True
```

```
b = False
```

```
c = a and b # c 會等於 False
```

```
d = a or b  # d 會等於 True
```

```
e = not b   # e 會等於 True
```

算術運算子 (arithmetic operator) 可用在整數 (integer) 及浮點數 (floating-point number) ，計算結果也為整數或浮點數

運算子	功能	範例
+	加	a + b
-	減	a - b
*	乘	a * b
/	除	a / b
%	取餘數	a % b
**	指數	a ** b

例如

```
a = 2
```

```
b = a + 2 # b 會等於 4
```

```
c = b - 2 # c 會等於 2
```

```
d = c * 2 # d 會等於 4
```

```
e = d / 2 # e 會等於 2.0

f = e % 2 # f 會等於 0.0

g = f ** 2 # g 會等於 0.0
```

+ 與 * 也可用於字串 (string) ，例如

```
a = "a"

b = a + "b" # 字串連接， b 會等於 "ab"

c = a * 2 # 字串重複， c 會等於 "aa"
```

相等性及關係運算子 (equality and relational operator) 的結果得到布林值 (Boolean value) ，不是 **True** 就是 **False**

運算子	功能	範例
==	相等	a == b
!=	不相等	a != b
>	大於	a > b
>=	大於等於	a >= b
<	小於	a < b
<=	小於等於	a <= b

例如

```
a = 11

b = 22

c = a >= b # c 會等於 False

d = a <= b # d 會等於 True
```

```
e = c == d # e 會等於 False
```

```
f = a != b # f 會等於 True
```

我們運用的等號，其實屬於指派運算子 (assignment operator) ，所謂的指派是把等號右邊的值給左邊的變數

運算子	功能	範例
=	指派	a = b
+=	相加同時指派	a += b
-=	相減同時指派	a -= b
*=	相乘同時指派	a *= b
/=	相除同時指派	a /= b
%=	取餘數同時指派 資料型態	a %= b
**=	取指數同時指派	a **= b

例如

```
a = 2
```

```
a += 2 # a 會等於 4
```

```
a -= 2 # a 會等於 2
```

```
a *= 2 # a 會等於 4
```

```
a /= 2 # a 會等於 2.0
```

```
a %= 2 # a 會等於 0.0
```

```
a **= 2 # a 會等於 0.0
```

也可以用平行指派 (parallel assignment)

```
a, b, c = 11, 22, 33

# a 會等於 11

# b 會等於 22

# c 會等於 33
```

有沒有看的眼花撩亂？居然有 f = a != b 之類的式子出來，這是因為運算子有計算的優先次序 (precedence) ，由於 != 的優先順序在 = 之前，所以 a != b 會先被計算，結果才會指派到 f 之中。如果不是很清楚各個運算子的優先次序，保險一點的做法可以用小括弧，先把要先被計算的運算式圍起來，例如

```
f = (a != b) # 等於 f = a != b
```

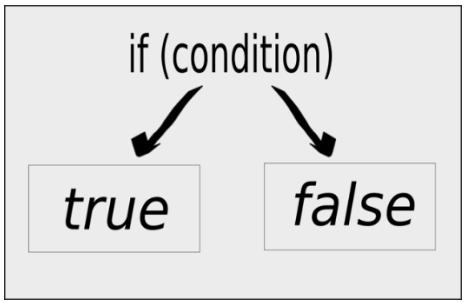
這是因為小括弧有最高的優先計算次序。

基本上運算式就是程式 (program) 中獨立的陳述 (statement) ，通常一行程式碼只會放一個陳述。 Python 程式的執行單位就是陳述，由前而後，一個陳述接著一個陳述來執行。

然而許多控制結構 (control structure) 是屬於多行的複合陳述 (compound statement) ，我們先來看看控制結構中的選擇吧！

2.5 選擇

程式中的選擇 (selection) 就是依條件 (condition) 使程式有不同的執行方向，若條件為真，也就是 *True* ，程式就會跳過 *False* 的部份執行 *True* 的部份，反之亦然



選擇結構有單一選擇跟多重選擇，兩者都使用 **if** 陳述 (if statement) ， **if** 為關鍵字 (keyword) 之一，若是多重選擇 **if** 須與 **elif** 或 **else** 連用。單一選擇，也就是單獨使用 **if** 陳述如下

```
if 3 > 5:
```

```
    print("Oh! 3 is bigger than 5!")
```

條件為 **if** 後面到冒號的範圍，上例為 3 大於 5 ，如果 3 大於 5 為真，程式就會執行條件後縮排的程式區塊 (block) ，其範圍一直到沒有縮排為止，這裡就可以看到 Python 裡頭縮排的意義。如果 3 大於 5 為假，程式自然跳過條件後的程式區塊，去找區塊後的第一個陳述 (statement) 執行。

if 與 **else** 連用，條件為真，執行 **if** 後的程式區塊，條件為假，就執行 **else** 後的程式區塊

```
if 3 > 5:
```

```
    print("Oh! 3 is bigger than 5!")
```

```
else:
```

```
    print("Not Bad! 3 is not bigger than 5!")
```

elif 表示其他的條件，形成 **if-elif-else** 的多重選擇，最後的 **else** 表示以上皆非

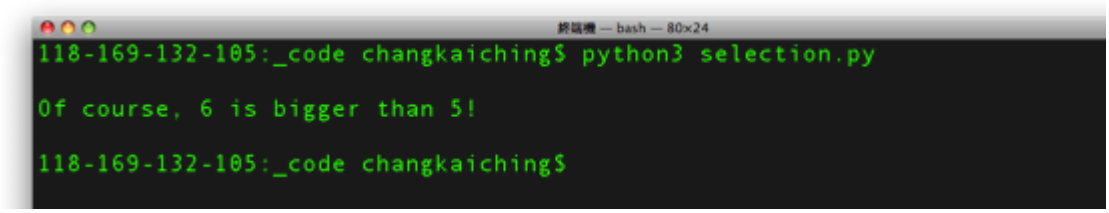
```
if 3 > 5:
    print("Oh! 3 is bigger than 5!")
elif 4 > 5:
    print("Oh! 4 is bigger than 5!")
elif 5 > 5:
    print("Oh! 5 is bigger than 5!")
elif 6 > 5:
    print("Of course, 6 is bigger than 5!")
else:
    print("There is no case :(")
```

將以上寫成完整的範例程式，如下

```
print()
if 3 > 5:
    print("Oh! 3 is bigger than 5!")
elif 4 > 5:
    print("Oh! 4 is bigger than 5!")
elif 5 > 5:
```

```
    print("Oh! 5 is bigger than 5!")
elif 6 > 5:
    print("Of course, 6 is bigger than 5!")
else:
    print("There is no case :(")
print()
```

執行結果如下

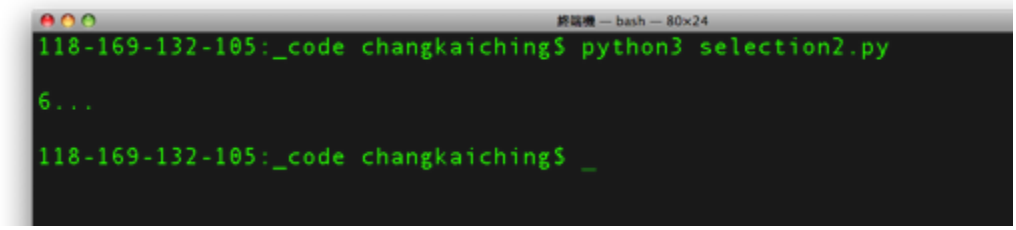


另舉一例如下

```
print()
s = 6
if s == 3:
    print("3...")
elif s == 4:
    print("4...")
elif s == 5:
    print("5...")
```

```
elif s == 6:
    print("6...")
else:
    print("There is no case :(")
print()
```

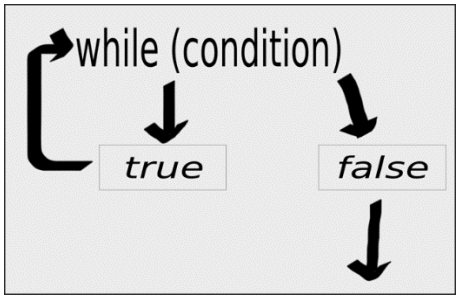
這裡的條件為判斷某一變數 (variable) 是否符合某一常數 (constant) ，執行結果如下



複合陳述 (compound statement) 除了選擇結構 (selection structure) 還有重複結構 (repetition structure) ，重複結構也被稱為迴圈 (loop) ，接下來我們就來看看如何使用迴圈吧！

2.6 迴圈

程式 (program) 中的迴圈 (loop) 就是在特定程式區塊 (block) 中，重複執行相同的工作



Python 中有兩種迴圈，分別是 **while** 迴圈 (while loop) 與 **for** 迴圈 (for loop) 。我們先來看看 **while** 迴圈的寫法

```
i = 10 # 設定控制變數

while i > 0:

    # 迴圈工作區

    print(i)

    i -= 1 # 調整控制變數值
```

這個迴圈所進行的工作很簡單，先在命令列上印出 10 ，然後一路遞減到 0 為止。迴圈有三個地方要注意

- 設定控制變數
- 條件
- 調整控制變數值

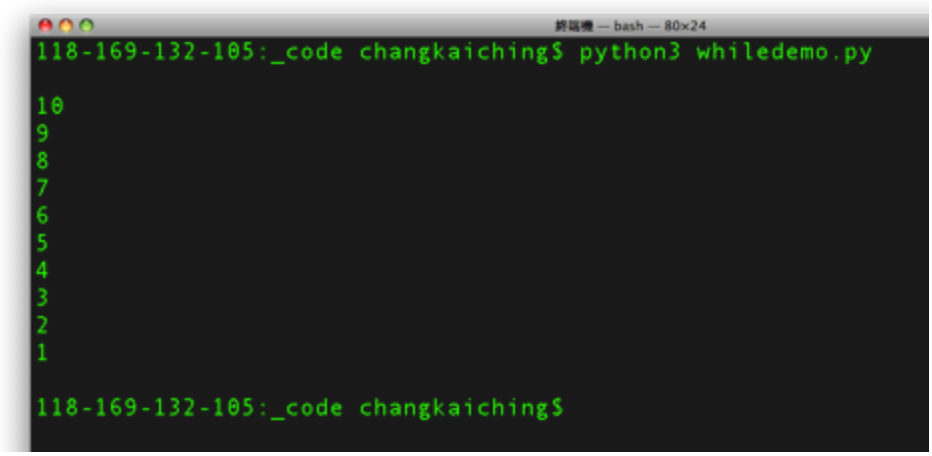
while 迴圈的控制變數 (control variable) 必須在 **while** 之前就先設定好，此例中將控制變數 i 設定為 10 。然後進入 **while** 的地方，條件 (condition) 就在 **while** 之後到冒號之前的地方，此例中為當控制變數 i 大於 0 時，迴圈便會重複執行。迴圈工作區，也就是 **while** 底下用縮排的程式區塊，這裡，我們只有簡單的印出控制變數 i 的值，迴圈工作區的最後需要有調整控制變數值的

地方。

先寫成完整的範例，來執行看看結果吧！

```
print()
i = 10 # 設定控制變數
while i > 0:
    # 迴圈工作區
    print(i)
    i -= 1 # 調整控制變數值
print()
```

執行結果如下



當我們明確知道重複次數的時候，我們得利用控制變數來記錄 **while** 迴圈所進行次數，這樣 **while** 迴圈才會有結束的一天，不然若是三個與控制變數相關的部份，漏了任一部份時，就有可能導致無窮迴圈 (infinite loop) 的發生，例如

```
i = 10 # 設定控制變數
```

```
while i > 0:
    # 迴圈工作區
    print(i)
```

這樣一來，控制變數 i 永遠大於 0 ，所以迴圈會一直重複執行，此例中會不斷的在命令列印出 10 ，直到強制結束程式的執行為止。

也可用 **break** 跳出迴圈， **contiune** 直接進行下一輪。

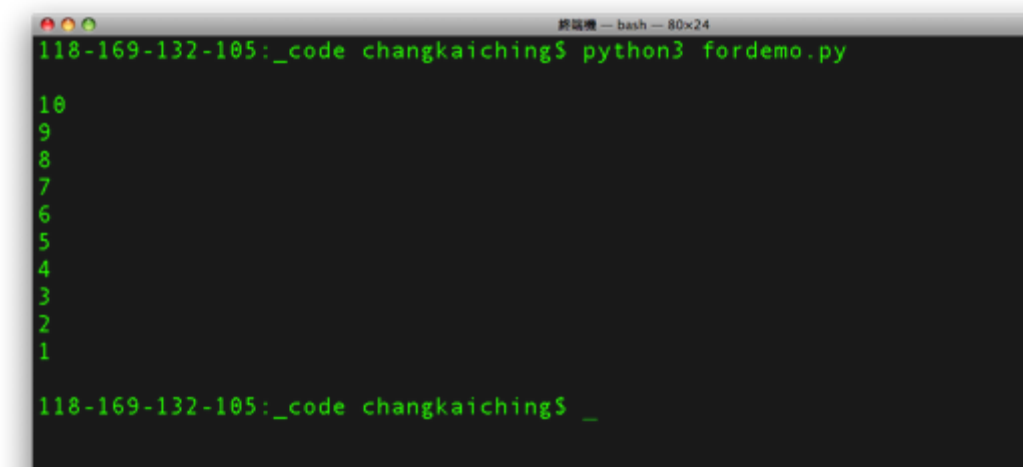
另外一個 **for** 迴圈用於取得具有多個元素的物件，例如內建函數 (function) range() 回傳一個依序的 range 物件

```
for i in range(10, 0, -1):
    print(i)
```

for 與 **in** 連用， **in** 後面接多個元素的物件。這個 **for** 迴圈與上面的 **while** 迴圈功能完全相同，寫成完整的範例程式，如下

```
print()
for i in range(10, 0, -1):
    print(i)
print()
```

此例的 range() 用了三個參數，第一個參數為起始值，第二個參數為結束值，第三個參數為遞增值。執行結果如下



雖然 **while** 迴圈與 **for** 迴圈可以互相替代，但還是依特性去使用比較適合囉！

接下來我們繼續看到可以將程式模組化的重要的概念，也就是定義自己的函數。

2.7 函數

函數 (function) 是一種功能性的單位，可以將程式 (program) 分割成小部分，藉由呼叫函數安排執行順序

定義函數使用關鍵字 (keyword) **def** ，其後空一格接函數名稱與小括弧，小括弧用來放參數列 (parameter list) ，函數可以有參數 (parameter) 也可以沒有參數，沒有參數的函數的小括弧留空，另外函數可用 **return** 設定回傳值 (return value) 。我們舉一例如下

```
def big(a, b):
    if a > b:
        return a
    else:
        return b
```

big() 函數回傳兩個參數中的較大值，參數數量則是依需求自己定義，這裡為兩個 *a* 與 *b* ，另外此例用了兩個 **return** ，這裡 **return** 就是函數結束執行，將控制權交還原本呼叫函數的地方。

將 big() 寫成一個完整範例

```
def big(a, b):
    if a > b:
```

```
        return a
    else:
        return b

print()
print(big(33, 22))
print(big("John", "Mary"))
print()
```

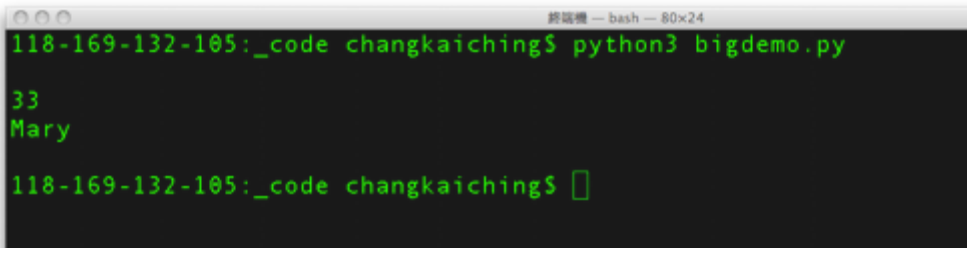
先比較 33 與 22 的大小，然後印出哪一個比較大

```
print(big(33, 22))
```

比較 "John" 與 "Mary" 的大小，字串比較是先比較第一個字元，按字母順序，順序越後面的值越大

```
print(big("John", "Mary"))
```

執行結果如下



函數常見的運用為重複性質的工作，我們另舉一例如下

```
def print_newline():
    print()
```

```
def print_something(a):
    print(a)

def return_something(a):
    return a

print_newline()
print_something(return_something("There is no spoon."))
print_newline()
```

這裡總共有三個函數， print_newline() 就是簡單的印出新行

```
def print_newline():
    print()
```

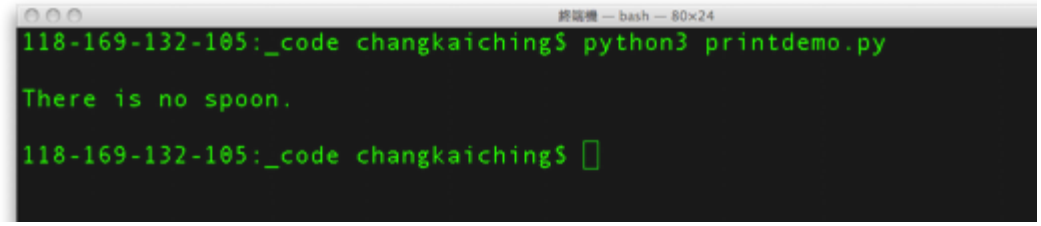
print_something() 則是印出參數 *a*

```
def print_something(a):
    print(a)
```

return_something() 則是回傳參數

```
def return_something(a):
    return a
```

執行結果如下



常見內建函式

除了自己可以建立函式外，Python 本身也提供許多好用的內建函式：

- all：列表中所有元素為真

```
all([True, 1, { 3 }]) # True
all([True, 1, { }]) # False
all([]) # True，沒有元素為假
```

- any：列表中只要有任何元素為真

```
any([True, 1, {}]) # True
any([]) # False，沒有元素為真
```

- enumerate：列舉

  我們常需要反覆取得列表中每個元素及其索引值

```
# choice 1
for i in range(len(documents)):
        document = documents[i]
        do_something(i, document)

# choice 2
i = 0
for document in documents:
        do_something(i, document)
        i += 1
```

  使用 enumerate：

```
for i, document in enumerate(documents):
       do_something(i, document)

# 僅需要 index
for i, _ in enumerate(documents): do_something(i)
```

- zip：合併將多個 list 合併成 tuple of list

```
list1 = ['a', 'b', 'c']
list2 = [1, 2, 3]
zip(list1, list2) # [('a', 1), ('b', 2)]

zip(*[('a', 1), ('b', 2), ('c', 3)])
    == zip(('a', 1), ('b', 2), ('c', 3))
       # [('a', 'b', 'c'), ('1', '2', '3')]
```

```
# * 可以參數拆分
def add(a, b): return a + b
add(1, 3) # 4
add([1, 3]) # TypeError
add(*[1, 3]) # 4
```

- range：取得一序列

```
range(0, 10) # 0, 1, 2, ... 9
```

- random：生成隨機數字

```
import random

randoms = [random.random() for _ in range(4)] # 生成長度為 4 內涵值為 0 - 1 不含 0 的 list
```

- 事實上，random 產生的是偽隨機數字，透過 seed 設定可以取得同樣值

```
import random
random.seed(10) # 把 seed 設為 10
print(random.random()) # 0.5714025946899135
random.seed(10) # 把 seed 設為 10
print(random.random()) # 0.5714025946899135
```

- 隨機產生範圍內數字：

```
random.randrange(10)
random.randrange(3, 6)
```

- 針對列表隨機排列：

```
num = range(10)
random.shuffle(num)
```

```
random.choice(['Mark', 'Bob', 'Jack'])
```

- 隨機取樣不放回：

```
nums = range(10)
random.sample(nums, 3)
```

- 隨機取樣放回：

```
nums = range(10)
random.choice(nums, 3)
```

- sort：針對 list 進行排序（由小到大），會改變原來的 list。sorted 不會改變原來 list

```
x = [4, 1, 2, 3]
y = sorted(x) # [1, 2, 3, 4] 不會改變到 x
x.sort() # x 變成 [1, 2, 3, 4]
```

- 若想改成由大到小排序

```
x = sorted([-4, 1, -2, 3, key=abs, reverse=True])
# [-4, 3, -3, 2] 絕對值由大到小
```

- 若是在 key 指定一個函數，就會用這個函數結果去做排序

```
wc = sorted(word_counts.items(),
key=lambda (word, count): count, reverse=True) # 針對單字數量多到小排序
```

- partial：使用函式工具創建另一個函式

```
from functools import partial
def exp(base, power):
       return base ** power
two_to_the = partial(exp, 2)
print_two_the(3)
```

- map：

```
def multiply(x, y):
	return x * y
map(multiply, [1, 2], [1, 2]) # [1, 4]
```

- filter：

```
def is_even(x):
	return x % 2 == 0
filter(is_even, [2, 5, 6]) # [2, 6]
```

- reduce：

```
def multiply(x, y):
	return x * y
reduce(multiply, [1, 2, 3]) # 1 * 2 * 3
```

2.8 控制流程

1. if…elif…else

```
if 1 > 2:
        message = 'if onlt 1 were greater than two'
elif 1 > 3:
        message = 'elif == else if'
else:
        message = 'else'
```

2. 三元運算子：

```
parity = 'even' if x % 2 == 0 else 'odd'
```

3. for…in

   較複雜情況我們會搭配 continue 和 break 使用：

```
for x in range(10): # 0...9 不含 10
        if x == 3:
                continue
        if x == 5:
                break
        print(x)
```

4. while

```
x = 0
while x < 10:
        print('x is less than 10')
        x += 1
```

生成器（generator）與迭代操作

事實上，list 有個問題就是很容易變得很大。例如：range(1000000) 就會製造出一個包含一百萬的元素的 list。若是想要使用其中幾個元素，效能就會變得很差。此時使用生成器（generator）就是一個每次只生成所需數值的一種 lazy 作法。

使用函式和 yield

```
def lazy_range(n):
	i = 0
	while i < n:
		yield i
		i += i
```

```
for i in lazy_range(10):
	do_something(i)
```

或是在小括號使用運算解析式

```
lazy_evens_below_20 = (i for i in lazy_range(20) if i % 2 == 0)

另外，每個 dict 都有一個叫做 items() 的方法

iteritems() # 可以一次生成一個鍵值對
```

裝飾器（decorator）

裝飾器本身是一個函式，主要是借助閉包力量產生一個可以修飾函式的函式：

```
@print_fun(title='title:')
def add(*tup):
	return sum(tup)

# 以下兩者相同
add(1, 2, 3, 4)
add = print_fun(title='title:')(add)

# 裝飾器撰寫
def print_fun(title):
	def decorator(func):
		def modified_func(*args, **kwargs):
```

```
                    result = func(*args, ** kwargs)
                    print(title, result)
            return modified_func
    return decorator
```

正規表達式

與許多程式語言一樣，Python 同樣提供正規表達式的用法，可以方便擷取文字。

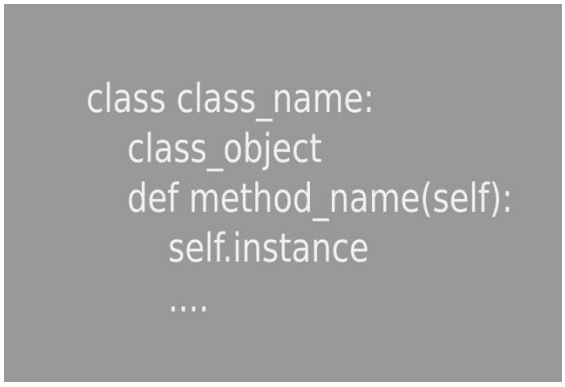
```
import re

print.all([
re.match('a', 'cat'),
re.search('a', 'cat')])
```

學會函數後，我們要進一步來看看怎麼設計類別 (class) 囉！

2.8 類別

類別 (class) 用來設計自己需要的物件 (object) ，這是說，類別是物件的模板。 Python 中設計類別使用關鍵字 (keyword) **class** ，裡頭可定義類別的類別屬性 (class attribute) 、實體屬性 (instance attribute) 與方法 (method)



舉一例如下

```
class Demo:

    def setAtt(self, a = 22, b = 33):

        self.a = a

        self.b = b

    def do_something(self):

        return self.a + self.b


print()

d = Demo()

d.setAtt()
```

```
print(d.do_something())
```

```
d.setAtt(11, 22)
```

```
print(d.do_something())
```

```
print()
```

此例中的 Demo 類別的 *setAtt* 方法定義兩個實體屬性 *self.a* 與 *self.b* ，亦有三個參數 *self* 、 *a* 與 *b* ，同時設定 *a* 的初值為 22 ， *b* 的初值為 33 ，然後將 *a* 設定給 *self.a* ， *b* 設定給 *self.b*

```
def setAtt(self, a = 22, b = 33):
```

```
    self.a = a
```

```
    self.b = b
```

self 為 Python 類別定義中預設的參數，代表建立的物件實體，因此 *self.a* 與 *self.b* 都是實體屬性。

另外定義一個 *do_something()* 方法，回傳 *self.a* 與 *self.b* 的相加值

```
def do_something(self):
```

```
    return self.a + self.b
```

建立新物件利用類別名稱的小括弧，然後呼叫 *do_something()* 兩次，第二次之前呼叫 *setAtt()* 重新設定 *self.a* 與 *self.b* 之值

```
print()
```

```
d = Demo()
```

```
d.setAtt()
```

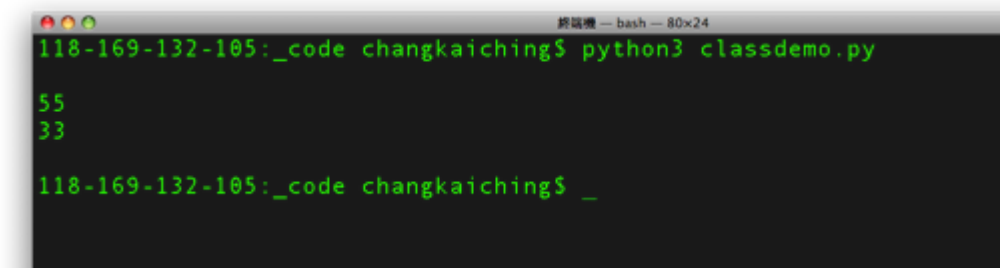
```
print(d.do_something())
```

```
d.setAtt(11, 22)
```

```
print(d.do_something())
```

```
print()
```

執行看看結果吧



物件導向程式設計（OOP）

Python 也是物件導向程式語言：

```
class Set:
	def __init__(self, values=None):
		# 建構函數
		s1 = Set()
		s2 = Set([1, 2, 3])
		self.dict = {}
		if values is not None:
			for value in values:
				self.add(value)
	def __repr__(self):
		return "Set" + str(self.dict.keys())
	def add(self, value):
		self.dict[value] = True
	def contains(self, value):
		return value in self.dict
```

```
        def remove(self, value):
                del self.dict[value]
# 使用物件
s = Set([1, 2, 3])

s.add(4)

print(s.contains(4))
```

例外狀況

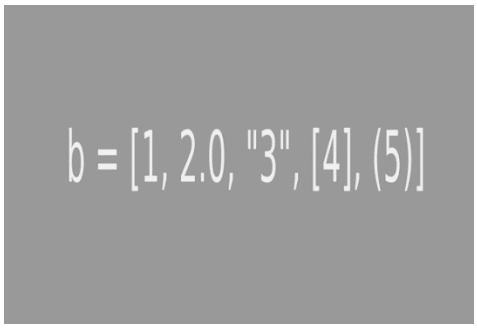
若是程式出現錯誤的話會送出 exception，若不妥善處理的話程式很有可能會掛掉，在 Python 中例外處理可以使用 try…except：

```
try:
        print(1/0)
except ZeroDivisionError:
        print('cannot divide by zero')
```

定義類別很簡單吧！我們將發展一個 Encrypt 類別，利用 Encrypt 物件進行編碼、解碼的工作，在此之前，我們先來想想密碼表該怎麼製作出來咧！這就需要用到串列 (list) 了。

2.9 串列

我們之前有簡單介紹過串列 (list) ，現在則是要進一步討論如何用串列製作密碼表



串列的字面常數 (literal) 為用中括弧圍起來的內容，同樣可以利用串列變數 (variable) 加中括弧存取裡頭的元素 (element) ，常見的操作如下表

操作	說明
d[i] = x	將索引值 i 的元素設定為 x
d[i:j] = t	將索引值 i 到 j 的元素設定為 t
del d[i:j]	刪除索引值 i 到 j 的元素
x in d	判斷 x 是否為 d 的元素
x not in d	判斷 x 是否非為 d 的元素
d + e	合併 d 與 e 兩個串列
d * 5	將 d 中所有元素複製為 5 倍

del 為關鍵字 (keyword) 之一，用來刪除物件 (object) 。

以下為串列的常用方法 (method)

方法	說明
d.append(x)	將 x 附加為 d 的最後一個元素
d.extend(L)	將 L 中的元素附加到 d 的最後
d.insert(i, x)	將 x 插入 d 索引值為 i 的地方
d.remove(x)	移除 d 中第一個 x 元素
d.pop([i])	取出 d 中索引值為 i 的元素，預設是最後一個
d.index(x)	取得 d 中第一次出現 x 的索引值
d.count(x)	累計 s 中 x 出現的個數
d.sort()	排序 d 中的元素

d.reverse()	倒轉 d 中元素的順序

串列還可以直接在中括弧中進行綜合運算 (comprehension) ，以此初始化串列元素，例如

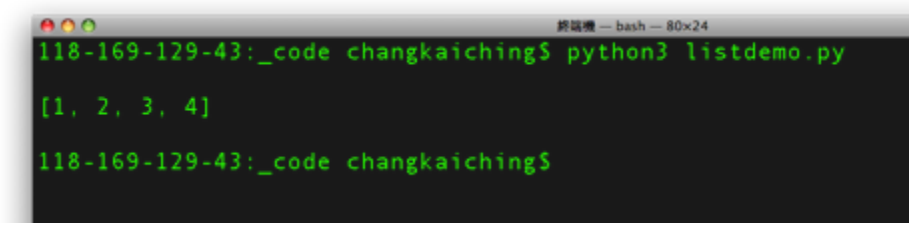
```
d = [i for i in range(1, 5)]

print()

print(d)

print()
```

這樣 *d* 就會得到 1 、 2 、 3 、 4 等四個數字，執行結果如下



同樣的方式，我們也可以製作出按順序的 26 個英文小寫字母表，例如

```
code = [chr(i) for i in range(97, 123)]

code_str = "".join(code)

print()

print(code_str)

print()
```

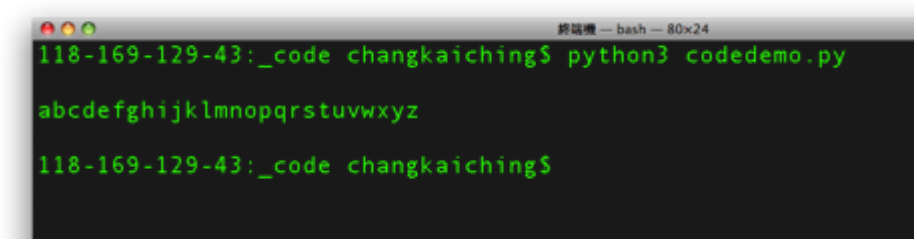
由於英文小寫字母 a 的編碼是 97 ，到 z 為 122 ，因此用 range() 取得 97 到 122 的整數，再用內建函數 chr() 將整數轉化為 Unicode 字元，這樣 code 中的元素就會是 "a" 、 "b" 然後一直到 "z"

```
code = [chr(i) for i in range(97, 123)]
```

然後我們用字串 (string) 的 join() 方法將 code 中的所有元素連接成一個字串

```
code_str = "".join(code)
```

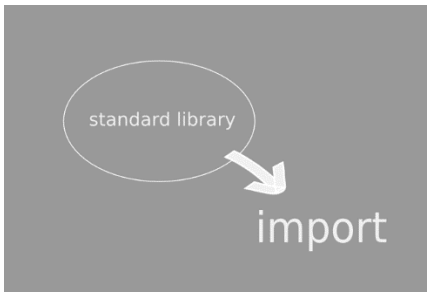
最後就是印出這個字串，來執行看看結果吧



可是這是按照順序的英文小寫字母表，如果要用作密碼表，還需要攪亂一下順序才行。標準模組庫中 random 模組的 shuffle() 剛好有這樣的功能，所以我們要先 **import** random 囉！

2.10 import

除了內建功能外， Python 的標準模組庫 (standard library) 還有許多已經定義好，並且測試無誤的模組 (module)



模組就是已經寫好的 Python 程式檔案，我們在需要的時候使用關鍵字 (keyword) **import** 到我們自己的程式中就可以使用相關定義，同樣的，使用標準模組庫中的模組也要 **import** ，例如下例 **import** 標準模組庫中的 random 模組

```
import random
```

```
code = [chr(i) for i in range(97, 123)]
```

```
random.shuffle(code)
```

```
code_str = "".join(code)
```

```
print()
```

```
print(code_str)
```

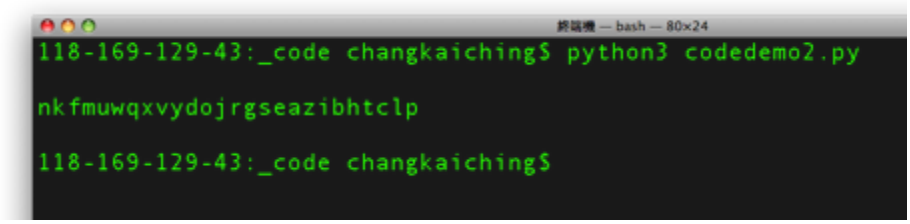
```
print()
```

這樣我們就可以使用 random 中的 shuffle() 函數 (function) ，把 *code* 攪亂一下裡頭的元素順序囉

```
random.shuffle(code)
```

import 模組後，使用模組內的定義需要連帶模組名稱，也可以用另一關鍵字 **form** 連用 **import** ，這樣就可以直接引入需要用的名稱。

執行結果如下



看起來還不錯，這就是把二十六個小寫英文字母
abcdefghijklmnopqrstuvwxyz

改變一下順序
nkfmuwqxvydojrgseazibhtclp

可藉由這個表格對英文句子中的小寫英文字母進行對換，例如 "There is no spoon." 可能變成以下任一個
Tfqdq ki jo itooj.
Tcnan hf gl fqllg.
Tczmz dn ij nkjji.
Tgfsf pb ir barri.
Tdcpc my fo yxoof.