

科學計算程式設計實習

Python 基本語法

王昱景 Brian Wang

brian.wang.frontline@gmail.com

何謂直譯式語言？

- 語言分為自然語言、低階語言和高階語言
- 自然語言就是人類在使用的語言，包含中文、英文、日文...等
- 機械語言跟組合語言合稱低階語言
- 使用 0 和 1 來編寫的程式語言叫機械語言，對人來說相當難閱讀



101 0111
110 1001
110 1011
110 1001
110 1001
111 0000
110 0101
110 0100
110 1001
110 0001

- 用簡單的字串和語法來取代全部使用 0 和 1 編寫的機械碼稱為組合語言
- 組合語言還需要經過組譯器將組合語言翻譯成機械碼，因結構及語法簡單速度快
- 低階語言在撰寫時很難進一步抽象化，對人類難閱讀，但對電腦來說好理解，執行速度快
- 高階語言讓人類寫程式方便，更接近自然語言

- 常見高階語言有 Python, Ruby, C, C++, Objective-C, C#, Java
- 高階語言又分直譯式語言、編譯式語言、混合語言
- 直譯式語言是在執行時利用直譯器，一邊將程式碼直譯為機械碼，同時另一邊執行該機器碼，也稱為腳本式語言 (Script Language)

- 直譯式語言具有動態、靈活的資料型別處理及轉換
- 因為直譯式語言是逐行直譯，因此通常速度比起編譯式語言來的慢
- 編譯式語言是在執行前會藉由編譯器將程式碼編譯成目的碼，再由連結器將其轉換為可執行的機械碼
- 因為在執行前會先編譯，所以對程式碼的要求就較為嚴謹而缺乏彈性但速度快

- 混合式語言則是兼具直譯式和編譯式兩者特性
- 混合式語言會先將程式碼藉由編譯器編譯成中介碼，然後等到要被執行時，再由該語言的專有執行環境進行即時編譯（Just In Time Compilation）而將中介碼編譯為機械碼以供電腦執行
- 混合式語言專有的執行環境就扮演直譯器角色（Java Virtual Machine, JVM / Common Language Runtime, CLR）

物件導向 (Object-Oriented, OO)

- 物件導向程式設計 (Object-Oriented-Programming, OOP) 是一種程式設計典範，同時也是一種程式開發的方法
- 物件指的是類別的例項
- 將物件作為程式的基本單元，將程式和資料封裝其中，以提高軟體的重用性、靈活性和擴充功能性

- 物件在程式中被看作一個實體，也就是家實際上使用的東西
- 除了具體可以看到的東西以外，其他抽象但也是被使用的東西也一樣能被稱為物件
- James Martin 及 James J. Odell 對物件的定義為”An Object is anything to which a concept applies. It is an instance of a concept.”

- 只要是依據特定概念所衍生出來的實體都可以稱為物件
- 物件導向程式設計三大特色：繼承、封裝及多型
- 根據 Deborah J. Armstrong 所提出的基本理論至少包含：類別、物件、訊息傳遞、繼承、多型、封裝這些特性

類別 (Class)

- 類別跟物件是密不可分的兩者
- 類別定義了物件的屬性與方法
- 根據類別實作出來的實體就是物件

類別 狗

開始

公有成員:

吠叫():

私有成員:

毛皮顏色:

孕育:

結束

物件 (Object)

- 實際做出一個物件就是像電腦的記憶體索取空間，用來記錄物件的屬性與方法
- 物件的屬性也可以被看作表示該物件的狀態
- 物件可以在建立時依據不同個體的需求給予不同的屬性，並且用不同的方式來使用相同的方法

定義萊絲是狗
萊絲.毛皮顏色:棕白色
萊絲.吠叫()

訊息傳遞 (Message Passing)

- 不同的類別所時做出來的物件並不一定只能單獨作業，時常需要跟其他物件相互合作
- 兩者之間需要相互傳遞資料 (訊息)

繼承 (Inheritance)

- 繼承就是一個類別 A 擁有子類別 B，此時類別 A 稱為類別 B 的父類別
- 子類別的目的是在於將父類別的定義更加精準化或者稱為抽象化、特殊化
- 兩個類別之間如果有繼承關係，如類別 A 被類別 B 繼承，則可以說類別 B 是類別 A 的一種 (Class B is a kind of Class A)

類別牧羊犬:繼承狗

定義萊絲是牧羊犬

萊絲.吠叫() /* 注意這裡呼叫的是狗這個類別的吠叫方法。*/

類別吉娃娃犬:繼承狗
開始

公有成員:
顫抖()

結束

類別牧羊犬:繼承狗

定義萊絲是牧羊犬

萊絲.顫抖() /* 錯誤：顫抖是吉娃娃犬的成員方法。 */

多型 (Polymorphism)

- 多型在OOP中不只一種意思
- 狹義來說多型就是所謂的同名異式，也就是利用動態繫結 (Dynamic Binding)，在程式執行的時候 (Run-time) 才動態決定該從多個相同名稱的方法 (函數) 中呼叫哪一個方法

- 而廣義來說函數的名稱共用（Overloading）以及函數的覆寫（Overriding）都算是多型
- Overloading 就是有多個不同的函數共用相同的名稱，差異點在於傳入函數的參數不同，來區分並呼叫不同的函數，以達到同名異式的目的，方便使用者在使用上的方便性以及減少函數的名稱

- **Overriding** 就是在子類別繼承父類別時，可以根據子類別的需求將父類別的方法及屬性改寫
- **Overriding** 的目的在於相同名稱的函數在不同類別中進行不同的動作


```
類別狗
開始
    公有成員:
        叫()
        開始
            吠叫()
        結束
    結束
```

```
類別雞
開始
    公有成員:
        叫()
        開始
            啼叫()
        結束
    結束
```

```
定義萊絲是狗
定義魯斯特是雞
萊絲.叫()
魯斯特.叫()
```

封裝 (Encapsulation)

- 封裝是保護程式內部資訊一種很重要的方式
- 在撰寫程式時如果輕易讓使用者獲得內部的資料，甚至可以自由更改內部設定或屬性，對於程式本身的安全性有相當大的威脅性
- 封裝可以讓使用者在不知道背後原理是如何運作的情形之下依舊使用該功能

- 對於使用者而言不用去在意背後原理讓使用者在操作上更輕鬆
- 對程式撰寫及維護人員而言可以保護程式中較重要的部分，避免讓使用者有心或無意的更改重要資訊，導致程式當掉或者更嚴重的資料遭到竊取或竄改

/* 一個程序導向的程式會這樣寫： */

定義萊絲

萊絲.設定音調(5)

萊絲.吸氣()

萊絲.吐氣()

/* 而當狗的吠叫被封裝到類別中，任何人都可以簡單地使用： */

定義萊絲是狗

萊絲.吠叫()

編碼

- 電腦無法理解人類所使用的自然語言，取而代之電腦可以理解的是數字的 0 和 1
- 舉凡所有數位的資訊都是由 0 和 1 所組合而成的，包括音訊、影像、文字全都是

- 電腦經過編碼這個動作把 0 跟 1 轉換成人類所看的懂得自然語言前面所提及的聲音與影像等
- 在所有程式語言中編碼都是相當重要的一環
- 如果編碼錯就會出現亂碼的文字

发件者: Microsoft Outlook Express Team 收件者: -s Outlook Express 用户
主题: 关于 Outlook Express 6

Outlook Express

用户

用户

- 关于 Outlook Express 6
- 关于 Outlook Express 6
- 关于 Outlook Express 6
- 关于 Outlook Express 6
- 关于 Outlook Express 6
- 关于 Outlook Express 6

关于 Outlook Express 6



Hotmail

关于 Outlook Express 6
关于 Outlook Express 6
关于 Outlook Express 6
关于 Outlook Express 6
关于 Outlook Express 6
关于 Outlook Express 6

- 在電腦裡面繁體中文編碼主要分成 BIG-5 以及 UTF-8
- 在 Python 的世界裡這兩種都支援
- BIG-5 是專門拿來作為繁體中文的編碼方式，對於其他語系的系統相容性較低
- UTF-8 是符合 Unicode 格式的變長度字元編碼

- 在 Python 3.x 版以後的版本也將 UTF-8 設為預設編碼
- #-*-coding:UTF-8-*-
- #coding:UTF-8
- #encoding:UTF-8
- #coding=UTF-8
- # coding:UTF-8

```
# coding=UTF-8  
text = '測試'  
print len(text) # 顯示 6
```

```
text = '測試'  
print(type(text)) # 顯示 "<class 'str'"  
print(len(text)) # 顯示 2
```

```
>>> '元'.encode('Big5')  
b'\xa4\xb8'  
>>> '元'.encode('UTF-8')  
b'\xe5\x85\x83'  
>>> '元'.encode('Big5').decode('Big5')  
'元'  
>>>
```

資料型態

- 在 Python 資料會以各種不同的型別儲存在電腦裡以供程式進行存取
- 主要可以分成三種類型：
 - 數值型別 (Numeric Type) int, float, bool, complex
 - 字串型別 (String Type) str
 - 容器型別 (Container Type) tuple, list, set, dict

數值型別

- `int` : `integer` 的縮寫，也就是整數的意思，負責儲存沒有小數點的整數數值
- `float` : 浮點數，可以儲存具有小數點的數值

- `bool`：boolean的縮寫，中文通稱“布林”，只有三種資料值，包括 `True`、`False` 與 `None`，分別代表真、假和空值三種意思
- `complex`：代表數學中的虛數，以 `complex(x, y)` 的形式出現，其中 `x` 為實部的數字，而 `y` 為虛部的數字
- 除此之外也可以在數字後面直接加 `j` 來代表虛數部，如：`1+5j`、`6.7-4.2j`

[illegible]

```
>>> 10 / 3
3.3333333333333335
>>> 10 // 3
3
>>> 10 / 3.0
3.3333333333333335
>>> 10 // 3.0
3.0
>>>
```


```
>>> 1.0 - 0.8
0.19999999999999996
>>> print (1.0 - 0.8)
0.2
>>>
```

```
>>> import decimal
>>> a = decimal.Decimal('1.0')
>>> b = decimal.Decimal('0.8')
>>> a - b
Decimal('0.2')
>>> print (a - b)
0.2
>>>
```

字串型別

- Python 可以用 `' '`、`""`、`''' '''` 把文字包住，
這些都代表字串的意思
- 字串有可以被拆解使用，且具有前後順序的特性
- `x="Hello,World"`
0 11



 brianwang — Python — 80x24

```
Last login: Wed Mar 11 05:16:59 on ttys000
Brian-Wang-MacBook:~ brianwang$ python
Python 2.7.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = "Hello, World"
>>> print(x[7])
W
>>> 
```

```
>>> 'c:\todo'
'c:\todo'
>>> print 'c:\todo'
c:  odo
>>> print 'c:\\todo'
c:\todo
>>>
```

可以在字串前加上 `r`，表示接下來後面是原始字串（Raw string）表示，這樣 Python 就會忠實表示後續的字串，技術上來說，會自動為你處理需要略過的字元

```
>>> r"c:\todo"
'c:\\todo'
>>> print(r"c:\todo")
c:\todo
>>> []
```

- Python 中的字串不可變 (Immutable) ，你無法改變已建立的字串內容
- 想得知字串實際的位元組序列長度 (不一定是字元長度) ，可以使用 len 函式
- 可以使用 for 來迭代字串
- 可以使用 in 來測試字串是否包括某子字串
- 可以使用 + 來串接字串
- 可以使用 * 來複製字串

```
>>> name = 'Justin'
>>> len(name)
6
>>> for ch in name:
...     print ch
...
J
u
s
t
i
n
>>> 'Just' in name
True
>>> name + name
'JustinJustin'
>>> name * 3
'JustinJustinJustin'
>>>
```

可以使用 `[]` 指定索引來取得字串中的某個字元，索引從 0 開始，可以是正數或負數，負數表示從尾端開始計數，例如 -1 就是最後一個字元，-2 就是倒數第二個字元，依此類推。例如：

```
>>> lang = 'Python'
>>> lang[0]
'p'
>>> lang[-1]
'n'
>>>
```

`[]` 也可以用來切割字串，例如：

```
>>> lang[1:5] # 取得索引 1 至 5（包括 1 但不包括 5）的子字串
'ytho'
>>> lang[0:] # 省略結尾索引，表示取至尾端
'Python'
>>> lang[:6] # 省略起始索引，表示從 0 開始
'Python'
>>>
```

`[]` 還可以指定間距（Gap），例如取索引 0 至 6，每 2 個間距的方式取子字串：

```
>>> lang[0:6:2]
'Pto'
>>>
```


'Python' 的 'P' 至 'y' 算一個間距，'y' 與 't' 之間也是一個間距，依此類推

'Python'[0:6:2] 取得的就是 'Pto'，將以上對 [] 的運算方式組合在一起，可以得到一個有趣的反轉字串方式[::-1]：

```
>>> lang[::-1]
'nohtyP'
>>>
```

```
>>> '{0} is {1}'.format('Justin', 'caterpillar')
'Justin is caterpillar'
>>> '{real} is {nick}'.format(real = 'Justin', nick = 'caterpillar')
'Justin is caterpillar'
>>> '{0} is {nick}'.format('Justin', nick = 'caterpillar')
'Justin is caterpillar'
>>> import sys
>>> 'My platform is {pc.platform}'.format(pc = sys)
'My platform is linux2'
>>>
```

容器型別

- 這個型別裡面的資料就像是箱子一樣，可以裝入各種不同型態的資料，也可以在箱子裡裝入其他的箱子
- **tuple**：序對，在 `()` 裡面可以放置一個以上的資料值，有順序性，但是不能更改其內容
- **list**：串列，在 `[]` 裡面可以放置一個以上的資料值，是一種有順序且可以更改其內容的型態

- **set**：集合，在 `{ }` 裡面可以放置一個以上的資料值，類似數學裡面的集合概念，所以內容並無順序性
- **dict**：字典，是 **dictionary** 的縮寫，以 **Key-Value** 對應的型態在 `{ }` 裡面放置一個以上的元素
- 字典是種配對型別，而 **key** 就是用來存取每個 **value** 的索引值

容器型別	tuple	list	set	dict
中文譯名	序對	串列	集合	字典
使用符號	()	[]	{}	{}
具順序性？	有	有	無	無
更改內容？	不可以	可以	可以	可以


```
Python 2.7.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 123
>>> print(type(x))
<type 'int'>
>>> x = '123'
>>> print(type(x))
<type 'str'>
>>> x = 123.0
>>> print(type(x))
<type 'float'>
>>> x = 123 + 0j
>>> print(type(x))
<type 'complex'>
>>> x = True
>>> print(type(x))
<type 'bool'>
>>> x = (123, 456)
>>> print(type(x))
<type 'tuple'>
>>> x = [123]
>>> print(type(x))
<type 'list'>
>>> x = {123}
>>> print(type(x))
<type 'set'>
>>> x = {1:123}
>>> print(type(x))
<type 'dict'>
```

`list` 與先前介紹過的字串型態享有共同的操作。`len` 傳回 `list` 長度；`in` 可測試某元素是否在 `list` 中；`+` 可以用來串接兩個 `list`；`*` 可用來複製出指定數量的 `list`。`[]` 可以指定索引，用以從 `list` 中取得元素，負索引是從最後一個元素計數，使用 `[]` 來切割 `list` 或許是最有用的功能。其他操作還有...

```
>>> [0] * 10
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> ', '.join(['justin', 'caterpillar', 'openhome'])
'justin, caterpillar, openhome'
>>> list('justin')
['j', 'u', 's', 't', 'i', 'n']
>>>
```

`set` 型態是無序群集，管理的元素不會重複而且是 `hashable`（這中間還有一些要討論的東西，可以參考 [物件相等性（上）](#)）。以下是 `set` 的幾個功能示範：

```
>>> admins = {'Justin', 'caterpillar'} # 建立 set
>>> users = {'momor', 'hamini', 'Justin'}
>>> 'Justin' in admins # 是否在站長群？
True
>>> admins & users      # 同時是站長群也是使用者群的？
{'Justin'}
>>> admins | users      # 是站長群或是使用者群的？
{'hamini', 'caterpillar', 'Justin', 'momor'}
>>> admins - users      # 站長群但不使用者群的？
{'caterpillar'}
>>> admins ^ users      # XOR
{'hamini', 'caterpillar', 'momor'}
>>> admins > users      # ∈
False
>>> admins < users
False
>>>
```

```
>>> passwords = {'Justin' : 123456, 'caterpillar' : 933933}
>>> passwords['Justin']
123456
>>> passwords['Hamimi'] = 970221    # 增加一對鍵值
>>> passwords
{'caterpillar': 933933, 'Hamimi': 970221, 'Justin': 123456}
>>> del passwords['caterpillar']    # 刪除一對鍵值
>>> passwords
{'Hamimi': 970221, 'Justin': 123456}
>>> passwords.items()
[('Hamimi', 970221), ('Justin', 123456)]
>>> passwords.keys()
['Hamimi', 'Justin']
>>> passwords.values()
```

```
[970221, 123456]
>>>
```

使用 `[]` 時如果指定的鍵不存在，會發生 `KeyError`，可以使用 `dict` 的 `get` 方法，指定鍵不存在時傳回的預設值。例如：

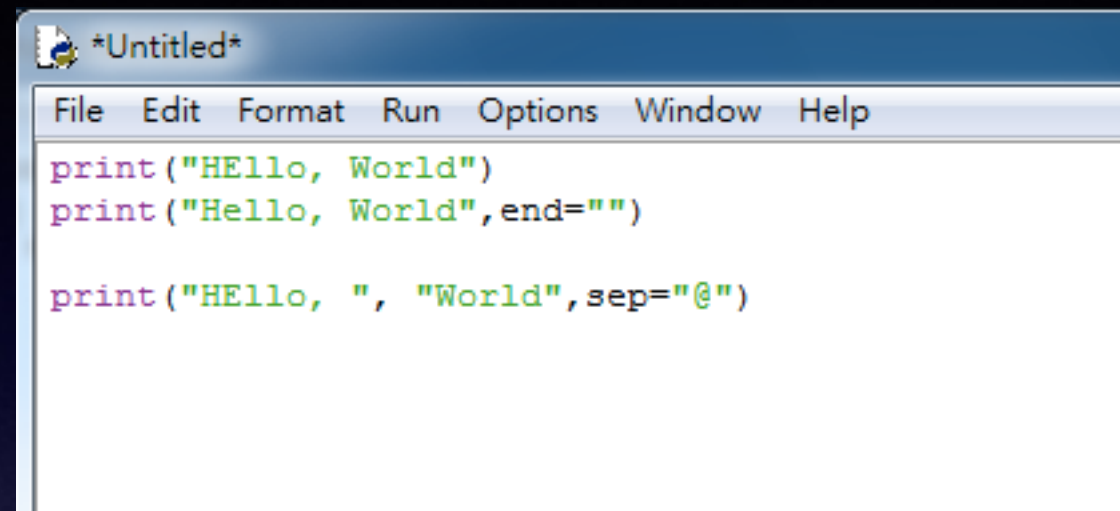
```
>>> passwords.get('openhome', '000000')
'000000'
>>> passwords['openhome']
Traceback (most recent call last):
  File "", line 1, in
KeyError: 'openhome'
>>>
```


輸出

- 在執行完程式碼中所執行的結果後將其輸出在螢幕上
- 在 3.x 版裡輸出的指令是“print()”，也就是英文的“印出”
- 在其後面接 () 表示此指令為一個函數
- () 裡面放想要輸出的文字

- `print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)`
- `value`：想要輸出的資料
- `...`：表可以用“,” 隔開多個想輸出的值
- `sep`：是將多個列印的值隔開的方式，預設為一個半形空白

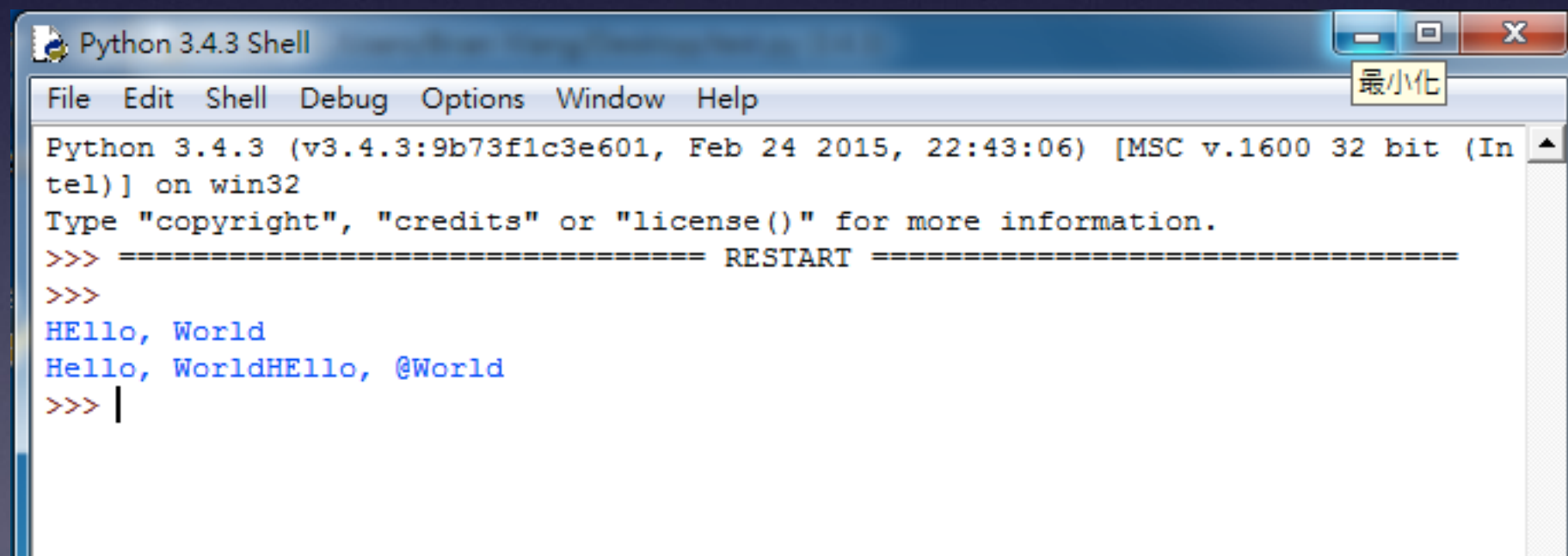
- `end`：是每一次 `print` 輸出之後會在最後面加上的內容，預設為 `"\n"` 也就是換行符號
- `file`：為輸出串流裝置，預設為 `sys.stdout`，也就是標準輸出裝置（螢幕）
- `flush`：是 3.3 版所增加的新功能，意思是在這行 `print` 指令執行後是否要強制將欲輸出之資訊輸出，而不是先放在緩衝區



```
*Untitled*
File Edit Format Run Options Window Help

print("HEllo, World")
print("Hello, World",end="")

print("HEllo, ", "World",sep="@")
```



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
HEllo, World
Hello, WorldHEllo, @World
>>> |
```


- 保留字（像 ‘、 “等）是不會被輸出至螢幕上
- 需要輸出這些保留字時需要加上 “\”

跳脫序列字元	說明	跳脫序列字元	說明
\a	響鈴	\v	垂直定位
\b	倒退	\\	印出反斜線
\f	換頁	\t	Tab鍵
\n	換行	\'	印出單引號
\r	歸位	\"	印出雙引號

```
print("testing")
print("print with reserved words including (\') (\") (\\) (\b)")
print("姓名\t年齡")
print("王小明\t16歲")
print("絕對無敵雷神王\t35歲",end="\n\n")
print("姓名\t\t年齡")
print("王小明\t\t16歲")
print("絕對無敵雷神王\t35歲")
```

輸入

- 如果一支程式只能輸出沒有辦法取得使用者所想要輸入的資料，程式彈性就會被限制縮小
- 在 Python 中若要取得使用者輸入的資訊使用的指令是“input()”，也就是英文的“輸入”
- 在 3.x 版 input() 所接受的值變成字串

- 如果需要取得數值，可以在取得 `input()` 的資料之後再轉型成數值
- 或者直接利用 `eval()` 將圓括號裡面的資料轉為數值型態，圓括號裡面的資料內容必須為數值型別的資料才行


```
string = input("現在輸入的是字串：")  
print(type(string))
```

```
integer1 = input("現在我想取得兩個數字來相除，先取分子：")  
integer2 = input("再取分母")  
print(type(integer1), type(integer2))
```

```
print(integer1/integer2)
```

```
integer1 = eval(input("重新要兩個數字："))  
integer2 = eval(input("取分母"))  
print(type(integer1), type(integer2))  
print(integer1/integer2)  
integer = eval(input("測試輸入非數值"))
```

註解

- “#” 後面所接的文字內容都是註解
- 還有三個單引號或三個雙引號也可以作為註解的表示
- 一般還說著解釋為了讓觀看程式碼的人能夠快速瞭解程式碼的目的、功能及使用方式

```
1  # A comment, this is so you can read your program later.
2  # Anything after the # is ignored by python.
3
4  print "I could have code like this." # and the comment after is ignored
5
6  # You can also use a comment to "disable" or comment out a piece of code:
7  # print "This won't run."
8
9  print "This will run."
```

```
$ python ex2.py
I could have code like this.
This will run.
```

縮排

- 相較於 Java 或 C 等程式語言，Python 有個相當大的差一點就是沒有用 {} 來當作區分程式碼的區塊，而是用縮排的方式來做區隔
- 縮排對於 Python 來說是相當重要的
- 如果縮排沒有對齊，那程式碼是無法執行的

- 在 Python 裡 Tab 鍵也可以拿來縮排
- 不過同時使用空白鍵與 Tab 鍵縮排需要特別注意
- 因為就算看起來縮排一致，但是空白鍵的縮排與 Tab 鍵的縮排計算方式是不一樣的

練習

習題1

請輸出以下詩 並利用"\t"與"\n"

>>>

登幽州臺歌

陳子昂

前不見古人

後不見來者

念天地之悠悠

獨愴然而涕下

>>>

習題2

請改寫以下程式碼，僅留下一行**print**指令且內容不使用空白字元，但仍維持一樣的輸出結果：

```
>>>
```

```
print(" 1")
```

```
print(" 2 3")
```

```
print(" 4 5 6")
```

```
print("7 8 9 10")
```

```
>>>
```

習題3

請撰寫一程式，讓使用者可以分別輸入2位同學的名稱及分數，輸出結果並如下排列：（斜體部分為執行程式時輸入）

```
>>>
```

請輸入A同學姓名:**Name_A**

請輸入A同學分數:**90**

請輸入B同學姓名:**Name_B**

請輸入B同學分數:**80**

姓名	分數
----	----

Name_A	90
---------------	-----------

Name_B	80
---------------	-----------

```
>>>
```

解答

```
print("登幽州臺歌\t陳子昂\n")    # \t為縮排(Tab)，\n為換行
print("前不見古人\t後不見來者\n念天地之悠悠\t獨愴然而涕下")
```

```
print("\b\b\b1\n\b\b2\b3\n\b4\b5\b6\n7\b8\b9\b10")    # \b為半形空白，\n為換行
```

```
nameA = input("請輸入A同學姓名:")    # 讓使用者輸入一段資料並存放至變數
nameA
scoreA = input("請輸入A同學分數:")    # 讓使用者輸入一段資料並存放至變數scoreA
nameB = input("請輸入B同學姓名:")    # 讓使用者輸入一段資料並存放至變數
nameB
scoreB = input("請輸入B同學分數:")    # 讓使用者輸入一段資料並存放至變數scoreB
print("姓名\t分數")
print(nameA + "\t" + scoreA)
print(nameB + "\t" + scoreB)
```