

**什麼是 Python？**

[Python](https://zh.wikipedia.org/wiki/Python) 是一種物件導向、直譯式的跨平台電腦程式語言，它包含了一組功能完備的標準庫和豐富套件生態系，可以輕鬆完成很多常見的任務（例如：讀寫檔案、自然語言處理、網路爬蟲、網站開發、機器學習等），因為它可以很輕易整合其他底層語言，所以又稱為膠水語言。它的語法簡單，與其它大多數程式設計語言使用大括弧不一樣，它使用縮進來定義語句塊。由於具備簡潔易學等特性，許多開發者推薦 Python 為初學者第一個學習的程式語言。由於版本更迭，我們接下來討論的主要是以 Python3 為主，若電腦沒有安裝的話，你可以在[官方網站下載](https://www.python.org/" \t "_blank)，若你不是安裝 [Anaconda](https://www.continuum.io/downloads) 這個 all-in-one 版本的話（自帶許多套件和科學運算工具），記得要安裝 [pip](https://pypi.python.org/pypi/pip)、[IPython](https://ipython.org/" \t "_blank)。

**Python 設計風格**

Python 主要設計的原則和特色就在於簡潔：應該會有一種明顯的作法（最好也只有一種），可以完成工作。

更多有關 Python 設計風格可以在終端機進入 python3 互動模式後輸入 import this：

|  |
| --- |
| $ python3 Python 3.5.2 (default, Oct 11 2016, 05:00:16) [GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darwin Type "help", "copyright", "credits" or "license" for more information. >>> import this  The Zen of Python, by Tim Peters  Beautiful is better than ugly. Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts. Special cases aren't special enough to break the rules. Although practicality beats purity. Errors should never pass silently. Unless explicitly silenced. In the face of ambiguity, refuse the temptation to guess. There should be one-- and preferably only one --obvious way to do it. Although that way may not be obvious at first unless you're Dutch. Now is better than never. Although never is often better than \*right\* now. If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea. Namespaces are one honking great idea -- let's do more of those! |

**空白格式**

首先，我們要了解 Python 和其他語言最大的不同就是使用縮排來切分程式碼，這和其他語言使用 {} 不同。

|  |
| --- |
| for i in [1, 2, 3]:  print(i)  for j in [1, 2, 3, 4]:  print(j + i)  print('finish') |

不過初學者很容易在縮排遇到問題，若是出現以下訊息就可以檢視是否哪裡縮排有問題：

|  |
| --- |
| IndentationError: expected an indented block |

**模組**

在 Python 生態系中有豐富的模組和工具。一般情況預設不會載入任何模組，但當你有特定開發需求可以使用第三方工具將模組匯入（import）。若是當模組名稱很長時通常我們會使用別名。

|  |
| --- |
| import re as regex  my\_regex = regex.compile('[0-9]+', regex.I) |

若是只是需要模組中的特定功能，也可以使用比較精準的引入方式 from import，引入到整個命名空間中，使用時前面就不用寫模組名（但要注意有可能覆寫）：

|  |
| --- |
| from collections import defaultdict, Counter lookup = defaultdict(int) my\_counter = Counter() |

**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

1. 數字（Number）

|  |
| --- |
| num1 = 3 num2 = 2 num3 = num1 / num2 |

1. 字串（String）  
   字串使用上會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) |

1. 若是多行的情形：

|  |
| --- |
| multi\_line\_str = """ 多行 多行 """ |

1. 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 |

1. 在 Python 中 None 地位類似於 null

|  |
| --- |
| x = None print(x == None) |

1. 以下為 Python 的 Falsy 值：
   * False
   * None
   * []
   * {}
   * “”
   * set()
   * 0
   * 0.0

可以搭配 and, or, not 使用

1. 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) |

1. 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 |

1. 切割（[起始 index, 結束 index 但不包含]）：

|  |
| --- |
| print(x[:3]) *# [-1, 1, 2]* print(x[3:]) *# [3, 4, 5,..., 9]* print(x[1:5]) *# [1, 2, 3, 4]* print(x[0:-1]) *# [1, 2, ..., 8]* print(x[-1:-1]) *# [-1, 1, ..., 9]* |

1. 檢查元素是否在列表中（逐一檢查，效率較差）：

|  |
| --- |
| 1 in [1, 2, 3] *# True* |

1. 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]* |

1. 賦值方式：

|  |
| --- |
| x, y = [1, 2] |
| \_, y = [1, 2] |

1. 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') |

1. 多重賦值

|  |
| --- |
| x, y = 1, 2 x, y = y, x # x == 2, y == 1 |

1. 字典（Dictionary）

字典類似 map，包含鍵值與對應的值，可以快速取出對應值：

|  |
| --- |
| dict1 = {} *# 建議寫法* dirct2 = dict() grades = { 'Mark': 70, 'Jack': 40 }  grades['Mark'] |

給定值：

|  |
| --- |
| grades['KD'] = 100 len(grades) *# 3* |

使用事先檢驗鍵或是使用 get 方法：

|  |
| --- |
| try:  grade = grades['XD'] except KeyError:  print('no grade for XD')  grades.get('XD', 40) *# 若無則使用 default 值* |

取出所有值：

|  |
| --- |
| grades = { 'Mark': 70, 'Jack': 40 } grades.keys() # 所有鍵值組成的 list grades.values() # 所有值組成的 list grades.items() # 所有鍵值組成的 tuple of list [('Mark', 70)] |

defaultdict：

當你檢查一個不存在的鍵值時，會用零參數函式添加一個事先設定的新值，這是一種比較優雅的作法

在介紹 defaultdict 之前先介紹一般作法

|  |
| --- |
| *# 例外處理* word\_counts = {} for word in document:  try:  word\_counts[word] += 1  except KeyError:  word\_counts[word] = 1 *# 使用 get* word\_counts = {} for word in document:  previous = word\_counts.get(word, 0)  word\_counts[word] = previous\_count + 1 |

defaultdict 作法（不用每次檢查鍵直視否存在）

|  |
| --- |
| *# 匯入 defaultdict* from collections import defaultdict  word\_counts = defaultdict(int) *# 會生成 0* for word in document:  word\_counts[word] += 1 |

也可以使用 list 或 dict 甚至是自己定義的函式來做為 defaultdict 的零參數函式：

|  |
| --- |
| dd\_list = defaultdict(list) dd\_list[1].append(2) *# { 1: [2] }*  dd\_list = defaultdict(list) dd\_list['Mark']['City'] = 'Bay Area' *# { 'Mark': { 'City': 'Bay Area'} }*  dd\_list = defaultdict(lambda: [0, 0]) dd\_pair[2][1] = 1 |

Counter：可以把一系列值轉成類似 defaultdict(int) 的東西，裡面每個值都對應到相應的數量，主要會使用來建立直方圖

|  |
| --- |
| from collections import Counter  c = Counter([0, 1, 2, 0]) *# { 0: 2, 1: 1, 2: 1 }*  word\_counts = Counter(document) |

每個 Counter 實例都有個 most\_common 的方法

|  |
| --- |
| for word, count in word\_counts.most\_common(10):  print(word, count) |

1. 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* |

1. 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}* lsit(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)] |

**函式**

函式（function）是重複使用的程式區塊，有輸入輸出。在 Python 中我們會使用 def 來定義函式：

|  |
| --- |
| def sum(x, y):  return x + y |

Python 的函數和 JavaScript 一樣，屬於一級函式（first-class）。我們可以將函數指定給某個變數，然後把它傳給某個函數中，就像是平常把參數傳入函式一樣。

|  |
| --- |
| def apply\_f(fun):  return fun(3, 2)  def sum(x, y):  return x + y   sum\_fun = sum num = apply\_f(sum) print(num) // 5 |

匿名函數使用方式（類似 ES6 arrow function 簡化寫法，前面是參數後面是操作）：

|  |
| --- |
| y = apply\_f(lambda x: x + 4) |

函式參數預設值：

|  |
| --- |
| def my\_print(message="default"):  print(message)  my\_print("hello python") my\_print() |

**args 與 kwargs**

若我們希望函式可以接受任意參數，我們可以使用 args（由無名稱參數組成的元組） 和 kwargs（由無名稱參數組成的字典）：

|  |
| --- |
| def magic(\*args, \*\*kwargs):  print('unnamed args:', args)  print('keywords args:', kwargs) magic(1, 2, key='word', key2='word2')   *# unnamed args: (1, 2)* *# keywords args: {'key': 'word', 'key2': 'word2'}* |

可以用來協助建立以函式當做參數的高階函式：

|  |
| --- |
| def double(f):  def g(\*args, \*\*kwargs):  return 2 \* f(\*args, \*\*kwargs)  return g  def f2(x, y):  return x + y  g = doubler\_correct(f2) print(g(1, 2)) |

**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 |

**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' |

1. 三元運算子：

|  |
| --- |
| parity = 'even' if x % 2 == 0 else 'odd' |

1. 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) |

1. 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')]) |

**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') |

**總結**

本文快速介紹了 Python 的基礎概念，當讀者學會了 Python 後，事實上可以嘗試使用 Python 開發不同的網路應用程式或是資料科學，甚至是自然語言處理的應用。