# Python技巧

**iterables**

根據Python標準文檔，Iterable的概念如下：

一種能夠一次返回一個成員的物件。

iterables的示例包括：

* 所有序列類型（如list、str和tuple）
* 一些非序列類型，如dict、檔物件以及類的實現中定義了\_\_iter\_\_()方法

Iterables是一個需要我們牢記的概念，因為接下來我們展示的許多技巧都使用itertools包。

itertools模組提供了一些函數，用於接收Iterable物件，而不僅僅是列印逐個對象。

## 1.

在工作學習中，我們經常會需要使用一個簡單的函數來實現從一個list來生成新的list, set或dict.此時我們就會用到iterables概念。

舉例來說：

生成List:

names = ['John', 'Bard', 'Jessica' 'Andres']

lower\_names = [name.lower() for name in names]

生成Set:

names = ['John', 'Bard', 'Jessica' 'Andres']

lower\_names = {

name.lower() for name in names}

生成Dict:

names = ['John', 'Bard', 'Jessica' 'Andres']

lower\_names = {

name:name.lower() for name in names}

個人建議：

僅當for語句、函式呼叫和方法調用的數量較少時使用。

## 2.

有時，我們需要獲得兩個清單物件之間的所有可能組合。

我們首先想到的實現可能如下：

l1 = [1, 2, 3]

l2 = [4, 5, 6]

combinations = []

for e1 in l1:

for e2 in l2:

combinations.append((e1, e2))

或者簡化一下，如下：

combinations = [(e1, e2) for e1 in l1 for e2 in l1]

上述實現已經很簡潔了，但標準庫itertools提供product函數，從而提供了相同的結果。如下所示：

from itertools import product

l1 = [1, 2, 3]

l2 = [4, 5, 6]

combinatios = product(l1, l2)

## 3.

假設有一個元素清單，我們需要在每對相鄰元素之間比較或應用一些操作，這有時稱為2個元素的滑動視窗。我們可以採用以下方式：

from itertools import tee

from typing import Iterable

def window2(iterable: Iterable):

it, offset = tee(iter(iterable))

next(offset)

return zip(it, offset)

l = [1, 2, 3, 4, 5, 6]

dd = window2(l)

for a in dd:

print(a)

運行結果如下：

(1, 2)

(2, 3)

(3, 4)

(4, 5)

(5, 6)

## 4.

有時，我們會需要一個類來存儲資訊，但是如果我們覺得創建一個類並定義其\_\_init\_\_()函數太麻煩時，我們不妨選擇使用dataclass。如下所示：

from dataclasses import dataclass

@dataclass

class Person:

name: str

age: int

address: str

上述代碼創建了一個具有預設構造函數的類，該類以與聲明相同的順序接收相應欄位的賦值。

person = Person(name='John', age=12, address='nanjing street')

dataclass的另一個優點是，預設情況下，會生成特殊方法，如\_\_str\_\_、 **repr** 、\_\_eq\_\_等。關於dataclass的更多用法，可以 [***參考官網***](https://link.zhihu.com/?target=https%3A//docs.python.org/3/library/dataclasses.html) 。

值得一提的是我們在類中聲明的成員變數的類型注釋（str、int等）並不強制在構造函數中傳遞的值屬於這種類型。也就是說dataclasses構造物件時並不執行資料類型的檢查。

## 5.

我們有時希望將一個物件上的操作視為tuple上的操作，一種選擇是使用collections.namedtuple,但也存在更類似於dataclass的實現。如下：

from typing import NamedTuple

class Coordinate(NamedTuple):

x: int

y: int

上述定義了一個標準的類可以被當做tuple來使用，如下：

coordinate = Coordinate(10, 15)

coordinate.x == coordinate[0] // True

coordinate.y == coordinate[1] // True

## 6.

假如我們有一個dataclass，需要驗證輸入資料是否符合類型注釋。在這種情況下，安裝協力廠商套裝軟體pydantic並將

*from dataclasses import dataclass* 替換為 *from pydantic.dataclasses import dataclass* 即可，如下：

from pydantic.dataclasses import dataclass

@dataclass

class Person:

name: str

age: int

address: str

這將生成一個類，該類具有根據成員變數聲明的類型進行輸入資料的解析和類型驗證。 *Pydantic* 在運行時強制執行類型提示，並在資料無效時提供友好的錯誤提醒。

## 7.

在某些情況下，我們需要生成一些容器中元素頻率的基本統計資訊。在這種情況下，您可以使用標準結構Counter來接收iterable並根據元素的頻率生成相應的統計資訊。

from collections import Counter

l = [1, 1, 2, 3, 4, 4]

frequencys = Counter(l)

print(frequencys[1]) // Ouput: 2

print(frequencys[2]) // Ouput: 1

print(frequencys[2323]) // Ouput: 0

Counter也提供了一些其他方法，比如如most\_common，用於檢索最常見的元素。

## 8.

如果我們相對兩個list中的元素對做相應的函數處理，我們最容易想到的方法如下：

l1 = [1, 2, 3]

l2 = [4, 5, 6]

for (e1, e2) in zip(l1, l2):

f(e1, e2)

但是使用函數map可以讓代碼更加簡潔一些。

l1 = [1, 2, 3]

l2 = [4, 5, 6]

map(f, l1, l2)

## 9.

有時候我們需要從一個list中隨機選擇一個元素，此時我們使用random.choice.如下所示：

from random import choice

l = [1, 2, 3]

random = choice(l)

如果我們需要隨機選擇多個元素呢？當然是使用random.choices.

from random import choices

l = [1, 2, 3, 4, 5]

random\_elements = choices(l, k=3)

上述代碼中的參數k為我們隨機選擇元素的個數。

* 字串操作
* 列表推導式
* lambda運算式
* 映射函數
* 單行實現if、elif和else條件判斷
* 拉鍊函數

## 1、字串操作

可以使用算數運算式來操作字串，例如 + 和 \* ：

>>> my\_string = "Hi Medium..!"

>>> print(my\_string \* 2)

Hi Medium..!Hi Medium..!

>>> print(my\_string + " I love Python" \* 2)

Hi Medium..! I love Python I love Python

也可以使用[::-1]來輕鬆地翻轉字串，而且不局限於字串！

>>> print(my\_string[::-1])

!..muideM iH>>> my\_list = [1,2,3,4,5]

>>> print(my\_list[::-1])

[5, 4, 3, 2, 1]

翻轉操作可以應用於單字清單嗎？當然！我們可以開發一個Yoda翻譯器：

>>> word\_list = ["awesome", "is", "this"]

>>> print(' '.join(word\_list[::-1]) + '!')

this is awesome!

上面我們使用.join()方法用空白字元號反向連接列表中的所有成員，最後再添加一個感嘆號。

## 2、列表推導式

你知道嗎，當我掌握這個技巧時，我感覺整個世界都變了！這種操作清單 的方式真的太強大、直觀而且可讀性高。

例如，假設我們有一個函數來計算輸入參數的平方再加5：

>>> def stupid\_func(x):

>>> return x\*\*2 + 5

現在我們要用這個函數處理一個清單中的所有奇數序號的成員，如果你不知道 列表推導式，可能會這樣來實現：

>>> my\_list = [1, 2, 3, 4, 5]

>>> new\_list = []

>>> for x in my\_list:

>>> if x % 2 != 0:

>>> new\_list.append(stupid\_func(x))

>>> print(new\_list)

[6, 14, 30]

但是有更簡單的實現方法！

>>> my\_list = [1, 2, 3, 4, 5]

>>> print([stupid\_func(x) for x in my\_list if x % 2 != 0])

[6, 14, 30]

清單推導式的語法是

[ expression for item in list ]

還可以使用額外的條件判斷運算式：

[ expression for item in list if conditional ]

上面這個運算式實際上等價於：

>>> for item in list:

>>> if conditional:

>>> expression

實際上你還可以進一步優化，因為我們其實不需要那個stupid\_func：

>>> print([x \*\* 2 + 5 for x in my\_list if x % 2 != 0])

[6, 14, 30]

## 3、Lambda運算式

Lambda看起來有點怪異，不過和這篇文章中的其他內容一樣，Lambda 超級強大，而且一旦你理解了就會發現它其實很直觀。

基本上Lambda運算式就是一個匿名函數。為什麼要匿名？這是因為Lambda 運算式通常應用於執行簡單的操作，因此使用def來正式定義一個函數 就顯得有點累贅了。

讓我們以上面的計算為例看一下Lambda的使用方法。在上面的代碼中 我們使用正式的函式宣告語法def stupid\_func(x)，現在讓我們使用 Lambda運算式：

>>> stupid\_func = (lambda x : x \*\* 2 + 5)

>>> print([stupid\_func(1), stupid\_func(3), stupid\_func(5)])

[6, 14, 30]

為什麼要使用這麼怪異的語法？Lambda的好處在於你可以執行一些簡單 的操作而無需提前定義一個函數。例如，假設有一組數值，我們如何 用Python來排序？一種方法是使用sorted()函數：

>>> my\_list = [2, 1, 0, -1, -2]

>>> print(sorted(my\_list))

[-2, -1, 0, 1, 2]

sorted()默認是按昇冪對列表成員排序，但是如果我們希望按成員的 平方和排序，就可以使用lambda運算式了：

>>> print(sorted(my\_list, key = lambda x : x \*\* 2))

[0, -1, 1, -2, 2]

## 4、映射函數 map

映射函數map()可以使用指定的函數來處理像清單之類的成員序列。例如， 假設我們希望得到兩個列表中對應成員的乘積的結果列表，應該怎麼做？ 使用Lambda運算式和映射函數就可以了！

>>> print(list(map(lambda x, y : x \* y, [1, 2, 3], [4, 5, 6])))

[4, 10, 18]

如果不使用Lambda和map的話，代碼就沒這麼優雅了：

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

>>> z = []

>>> for i in range(len(x)):

>>> z.append(x[i] \* y[i])

>>> print(z)

[4, 10, 18]

## 5、單行條件判斷

在你的代碼中可能會有這樣的條件判斷邏輯：

>>> x = int(input())

>>> if x >= 10:

>>> print("Horse")

>>> elif 1 < x < 10:

>>> print("Duck")

>>> else:

>>> print("Baguette")

運行上面的代碼，會提示我們輸入，假設輸入5的話，就會顯示 Duck。但是我們可以只用一行代碼就實現同樣的功能！

print("Horse" if x >= 10 else "Duck" if 1 < x < 10 else "Baguette")

實在是妙！

## 6、拉鍊函數 zip

上面的映射函數map()對兩個列表中並行地執行某個指定的處理函數。 拉鍊函數zip()比這還要簡單。

假設我們有兩個清單，一個記錄人名，另一個記錄姓氏，那麼我們應該 如何合併得到一個全名列表？答案是使用zip()！

>>> first\_names = ["Peter", "Christian", "Klaus"]

>>> last\_names = ["Nistrup", "Smith", "Jensen"]

>>> print([' '.join(x) for x in zip(first\_names, last\_names)])

['Peter Nistrup', 'Christian Smith', 'Klaus Jensen']

**1、for迴圈中的else條件**

這是一個for-else方法，迴圈遍歷列表時使用else語句。

下面舉個例子，比如我們想檢查一個列表中是否包含奇數。

那麼可以通過for迴圈，遍歷查找。

numbers = [2, 4, 6, 8, 1]

for number in numbers:

if number % 2 == 1:

print(number)

break

else:

print("No odd numbers")

如果找到了奇數，就會列印該數值，並且執行break語句，跳過else語句。

沒有的話，就不會執行break語句，而是執行else語句。

**▍2、從清單中獲取元素，定義多個變數**

my\_list = [1, 2, 3, 4, 5]

one, two, three, four, five = my\_list

**▍3、使用heapq模組，獲取清單中n個最大或最小的元素**

import heapq

scores = [51, 33, 64, 87, 91, 75, 15, 49, 33, 82]

print(heapq.nlargest(3, scores)) # [91, 87, 82]

print(heapq.nsmallest(5, scores)) # [15, 33, 33, 49, 51]

**▍4、將清單中的所有元素作為參數傳遞給函數**

我們可以使用 \* 號，提取清單中所有的元素

my\_list = [1, 2, 3, 4]

print(my\_list) # [1, 2, 3, 4]

print(\*my\_list) # 1 2 3 4

如此便可以將清單中的所有元素，作為參數傳遞給函數

def sum\_of\_elements(\*arg):

total = 0

for i in arg:

total += i

return total

result = sum\_of\_elements(\*[1, 2, 3, 4])

print(result) # 10

**▍5、獲取清單的所有中間元素**

\_, \*elements\_in\_the\_middle, \_ = [1, 2, 3, 4, 5, 6, 7, 8]

print(elements\_in\_the\_middle) # [2, 3, 4, 5, 6, 7]

**▍6、使用一行代碼賦值多個變數**

one, two, three, four = 1, 2, 3, 4

**▍7、列表推導式**

只用一行代碼，便可完成對陣列的反覆運算以及運算。

比如，將清單中的每個數字提高一倍。

numbers = [1, 2, 3, 4, 5]

squared\_numbers = [num \* num for num in numbers]

print(squared\_numbers) # [1, 4, 9, 16, 25]

推導式不僅清單能用，字典、集合、生成器也能使用。

下面看一下，使用字典推導式，將字典的值提高一倍。

dictionary = {'a': 4, 'b': 5}

squared\_dictionary = {key: num \* num for (key, num) in dictionary.items()}

print(squared\_dictionary) # {'a': 16, 'b': 25}

**▍8、通過Enum枚舉同一標籤或一系列常量的集合**

枚舉是綁定到唯一的常量值的一組符號名稱(成員)。

在枚舉中，成員可以通過身份進行比較，枚舉本身可以反覆運算。

from enum import Enum

class Status(Enum):

NO\_STATUS = -1

NOT\_STARTED = 0

IN\_PROGRESS = 1

COMPLETED = 2

print(Status.IN\_PROGRESS.name) # IN\_PROGRESS

print(Status.COMPLETED.value) # 2

**▍9、重複字串**

name = "Banana"

print(name \* 4) # BananaBananaBananaBanana

**▍10、比較3個數字的大小**

如果想比較一個值和其他兩個值的大小情況，你可以使用簡單的數學運算式。

1 < x < 10

這個是最簡單的代數運算式，在Python中也是可以使用的。

x = 3

print(1 < x < 10) # True

print(1 < x and x < 10) # True

**▍11、使用1行代碼合併字典**

first\_dictionary = {'name': 'Fan', 'location': 'Guangzhou'}

second\_dictionary = {'name': 'Fan', 'surname': 'Xiao', 'location': 'Guangdong, Guangzhou'}

result = first\_dictionary | second\_dictionary

print(result)

# {'name': 'Fan', 'location': 'Guangdong, Guangzhou', 'surname': 'Xiao'}

**▍12、查找元組中元素的索引**

books = ('Atomic habits', 'Ego is the enemy', 'Outliers', 'Mastery')

print(books.index('Mastery')) # 3

**▍13、將字串轉換為字串清單**

假設你在函數中獲得輸出，原本應該是一個列表，但實際上卻是一個字串。

input = "[1,2,3]"

你可能第一時間會想到使用索引或者規則運算式。

實際上，使用ast模組的literal\_eval方法就能搞定。

import ast

def string\_to\_list(string):

return ast.literal\_eval(string)

string = "[1, 2, 3]"

my\_list = string\_to\_list(string)

print(my\_list) # [1, 2, 3]

string = "[[1, 2, 3],[4, 5, 6]]"

my\_list = string\_to\_list(string)

print(my\_list) # [[1, 2, 3], [4, 5, 6]]

**▍14、計算兩數差值**

計算出2個數字之間的差值。

def subtract(a, b):

return a - b

print((subtract(1, 3))) # -2

print((subtract(3, 1))) # 2

上面的這個方法，需要考慮數值的先後順序。

def subtract(a, b):

return a - b

print((subtract(a=1, b=3))) # -2

print((subtract(b=3, a=1))) # -2

使用具名引數，安排順序，這樣就不會出錯了。

**▍15、用一個print()語句列印多個元素**

print(1, 2, 3, "a", "z", "this is here", "here is something else")

**▍16、在同一行列印多個元素**

print("Hello", end="")

print("World") # HelloWorld

print("Hello", end=" ")

print("World") # Hello World

print('words', 'with', 'commas', 'in', 'between', sep=', ')

# words, with, commas, in, between

**▍17、列印多個值，在每個值之間使用自訂分隔符號**

print("29", "01", "2022", sep="/") # 29/01/2022

print("name", "domain.com", sep="@") # name@domain.com

**▍18、不能在變數名的開頭使用數位**

four\_letters = "abcd" # this works

4\_letters = "abcd" # this doesn’t work

這是Python的變數命名規則。

**▍19、不能在變數名的開頭使用運算子**

+variable = "abcd" # this doesn’t work

**▍20、數字的第一位不能是0**

number = 0110 # this doesn't work

這個確實挺神奇的。

**▍21、在變數名的任何地方使用底線**

a\_\_\_\_\_\_b = "abcd" # this works

\_a\_b\_c\_d = "abcd" # this also works

這並不意味著，你可以無限使用，為了代碼的易讀性，還是需要合理使用。

**▍22、使用底線分割數值較大的數位**

print(1\_000\_000\_000) # 1000000000

print(1\_234\_567) # 1234567

如此，看到一大堆數字時，也能輕鬆閱讀。

**▍23、反轉列表**

my\_list = ['a', 'b', 'c', 'd']

my\_list.reverse()

print(my\_list) # ['d', 'c', 'b', 'a']

**▍24、使用步進函數對字串切片**

my\_string = "This is just a sentence"

print(my\_string[0:5]) # This

# Take three steps forward

print(my\_string[0:10:3]) # Tsse

**▍25、反向切片**

my\_string = "This is just a sentence"

print(my\_string[10:0:-1]) # suj si sih

# Take two steps forward

print(my\_string[10:0:-2]) # sjs i

**▍26、使用開始或結束索引進行切片**

my\_string = "This is just a sentence"

print(my\_string[4:]) # is just a sentence

print(my\_string[:3]) # Thi

**▍27、/和//的區別**

print(3/2) # 1.5

print(3//2) # 1

**▍28、==和is的區別**

is：檢查兩個變數是否指向同一物件記憶體中

==：比較兩個物件的值

first\_list = [1, 2, 3]

second\_list = [1, 2, 3]

# 比較兩個值

print(first\_list == second\_list) # True

# 是否指向同一記憶體

print(first\_list is second\_list)

# False

third\_list = first\_list

print(third\_list is first\_list)

# True

**▍29、合併字典**

dictionary\_one = {"a": 1, "b": 2}

dictionary\_two = {"c": 3, "d": 4}

merged = {\*\*dictionary\_one, \*\*dictionary\_two}

print(merged) # {'a': 1, 'b': 2, 'c': 3, 'd': 4}

**▍30、檢查字串是否大於另一字串**

first = "abc"

second = "def"

print(first < second) # True

second = "ab"

print(first < second) # False

**▍31、檢查字串是否以特定字元開頭(不使用索引)**

my\_string = "abcdef"

print(my\_string.startswith("b")) # False

**▍32、使用id()查找變數的唯一id**

print(id(1)) # 4325776624

print(id(2)) # 4325776656

print(id("string")) # 4327978288

**▍33、整數、浮點數、字串、布林值和元組都是不可變的**

當變數被賦值為整數、浮點數、字串、布林值、元組這些不可變類型後，該變數就會指向一個記憶體物件。

如果重新給變數再賦值，它的記憶體物件就會發生改變。

number = 1

print(id(number)) # 4325215472

print(id(1)) # 4325215472

number = 3

print(id(number)) # 4325215536

print(id(1)) # 4325215472

**▍34、字串和元組也是不可變的**

此處再說明一次。

name = "Fatos"

print(id(name)) # 4422282544

name = "fatos"

print(id(name)) # 4422346608

**▍35、清單、集合和字典都是可變的**

這意味著發生更改時，不會改變其記憶體物件。

cities = ["Beijing", "Guangzhou", "chengdu"]

print(id(cities)) # 4482699712

cities.append("Beijing")

print(id(cities)) # 4482699712

下麵是字典。

my\_set = {1, 2, 3, 4}

print(id(my\_set)) # 4352726176

my\_set.add(5)

print(id(my\_set)) # 4352726176

**▍36、把一個列表變成不可變的列表**

my\_set = frozenset(['a', 'b', 'c', 'd'])

my\_set.add("a")

使用frozenset()後，你就無法更改了。

**▍37、if-elif塊可以在沒有else塊的情況下存在**

但是elif不能在沒有if語句之前獨立存在。

def check\_number(number):

if number > 0:

return "Positive"

elif number == 0:

return "Zero"

return "Negative"

print(check\_number(1)) # Positive

**▍38、使用sorted()檢查2個字串是否為相同**

def check\_if\_anagram(first\_word, second\_word):

first\_word = first\_word.lower()

second\_word = second\_word.lower()

return sorted(first\_word) == sorted(second\_word)

print(check\_if\_anagram("testinG", "Testing")) # True

print(check\_if\_anagram("Here", "Rehe")) # True

print(check\_if\_anagram("Know", "Now")) # False

**▍39、獲取字元的Unicode值**

print(ord("A")) # 65

print(ord("B")) # 66

print(ord("C")) # 66

print(ord("a")) # 97

**▍40、獲取字典的鍵**

dictionary = {"a": 1, "b": 2, "c": 3}

keys = dictionary.keys()

print(list(keys)) # ['a', 'b', 'c']

**▍41、獲取字典的值**

dictionary = {"a": 1, "b": 2, "c": 3}

values = dictionary.values()

print(list(values)) # [1, 2, 3]

**▍42、交換字典的鍵、值位置**

dictionary = {"a": 1, "b": 2, "c": 3}

reversed\_dictionary = {j: i for i, j in dictionary.items()}

print(reversed) # {1: 'a', 2: 'b', 3: 'c'}

**▍43、將布林值轉換為數位**

print(int(False)) # 0

print(float(True)) # 1.0

**▍44、在算數運算中使用布林值**

x = 10

y = 12

result = (x - False)/(y \* True)

print(result) # 0.8333333333333334

**▍45、將任何資料類型轉換為布林值**

print(bool(.0)) # False

print(bool(3)) # True

print(bool("-")) # True

print(bool("string")) # True

print(bool(" ")) # True

**▍46、將值轉換為複數**

print(complex(10, 2)) # (10+2j)

也可以將數位轉換為十六進位數。

print(hex(11)) # 0xb

**▍47、在列表的第一個位置添加一個值**

如果使用append()，將從列表的最後一個位置插入新值。

可以通過使用insert()，來指定插入新元素的索引和數值。

那麼列表的第一個位置為0，即下標為0。

my\_list = [3, 4, 5]

my\_list.append(6)

my\_list.insert(0, 2)

print(my\_list) # [2, 3, 4, 5, 6]

**▍48、Lambda函數只能在一行代碼中**

無法通過多行代碼，來使用lambda函數。

comparison = lambda x: if x > 3:

print("x > 3")

else:

print("x is not greater than 3")

報錯。

**▍49、Lambda中的條件陳述式應始終包含else語句**

comparison = lambda x: "x > 3" if x > 3

運行上面的代碼，報錯。

這是由於條件運算式的特性，而不是lambda的導致的。

**▍50、使用filter()，獲得一個新物件**

my\_list = [1, 2, 3, 4]

odd = filter(lambda x: x % 2 == 1, my\_list)

print(list(odd)) # [1, 3]

print(my\_list) # [1, 2, 3, 4]

**▍51、map()返回一個新物件**

map()函數將給定函數應用於可反覆運算物件(清單、元組等)，然後返回結果(map對象)。

my\_list = [1, 2, 3, 4]

squared = map(lambda x: x \*\* 2, my\_list)

print(list(squared)) # [1, 4, 9, 16]

print(my\_list) # [1, 2, 3, 4]

**▍52、range()的step參數**

for number in range(1, 10, 3):

print(number, end=" ")

# 1 4 7

**▍53、range()默認從0開始**

def range\_with\_zero(number):

for i in range(0, number):

print(i, end=' ')

def range\_with\_no\_zero(number):

for i in range(number):

print(i, end=' ')

range\_with\_zero(3) # 0 1 2

range\_with\_no\_zero(3) # 0 1 2

**▍54、不需要和0比較長度**

如果長度大於0，則默認為True。

def get\_element\_with\_comparison(my\_list):

if len(my\_list) > 0:

return my\_list[0]

def get\_first\_element(my\_list):

if len(my\_list):

return my\_list[0]

elements = [1, 2, 3, 4]

first\_result = get\_element\_with\_comparison(elements)

second\_result = get\_element\_with\_comparison(elements)

print(first\_result == second\_result) # True

**▍55、可以在同一個作用域內多次定義一個方法**

但是，只有最後一個會被調用，覆蓋以前。

def get\_address():

return "First address"

def get\_address():

return "Second address"

def get\_address():

return "Third address"

print(get\_address()) # Third address

**▍56、在外部直接訪問私有屬性**

在定義屬性或方法時，在屬性名或者方法名前增加兩個底線，定義的就是私有屬性或方法。

如果想要在外部訪問，那麼只需要在名稱前面加上 '\_類名' 變成 '\_類名\_\_名稱'。

class Engineer:

def \_\_init\_\_(self, name):

self.name = name

self.\_\_starting\_salary = 62000

dain = Engineer('Dain')

print(dain.\_Engineer\_\_starting\_salary) # 62000

**▍57、檢查物件的記憶體使用情況**

import sys

print(sys.getsizeof("bitcoin")) # 56

**▍58、定義一個方法，可以調用任意個參數**

def get\_sum(\*arguments):

result = 0

for i in arguments:

result += i

return result

print(get\_sum(1, 2, 3)) # 6

print(get\_sum(1, 2, 3, 4, 5)) # 15

print(get\_sum(1, 2, 3, 4, 5, 6, 7)) # 28

**▍59、使用super()或父類的名稱調用父類的初始化**

使用super函式呼叫父類的初始化方法。

class Parent:

def \_\_init\_\_(self, city, address):

self.city = city

self.address = address

class Child(Parent):

def \_\_init\_\_(self, city, address, university):

super().\_\_init\_\_(city, address)

self.university = university

child = Child('Peking University', 'Fudan University', 'Tsinghua University')

print(child.university) # Tsinghua University

使用父類的名稱調用父類。

class Parent:

def \_\_init\_\_(self, city, address):

self.city = city

self.address = address

class Child(Parent):

def \_\_init\_\_(self, city, address, university):

Parent.\_\_init\_\_(self, city, address)

self.university = university

child = Child('Peking University', 'Fudan University', 'Tsinghua University')

print(child.university) # Tsinghua University

**▍60、在類中使用 + 操作符**

在兩個int資料類型之間使用 + 運算子時，將得到它們的和。

而在兩個字串資料型別之間使用它時，會將其合併。

print(10 + 1) # 兩數相加

print('first' + 'second') # 字串相加

這個就是操作符重載，你還可以在類中使用(\_\_add\_\_)。

class Expenses:

def \_\_init\_\_(self, rent, groceries):

self.rent = rent

self.groceries = groceries

def \_\_add\_\_(self, other):

return Expenses(self.rent + other.rent,

self.groceries + other.groceries)

april\_expenses = Expenses(1000, 200)

may\_expenses = Expenses(1000, 300)

total\_expenses = april\_expenses + may\_expenses

print(total\_expenses.rent) # 2000

print(total\_expenses.groceries) # 500

**▍61、在類中使用 < 和 == 操作符**

下面定義一個操作重載示例( < 操作符)，使用\_\_lt\_\_方法。

class Game:

def \_\_init\_\_(self, score):

self.score = score

def \_\_lt\_\_(self, other):

return self.score < other.score

first = Game(1)

second = Game(2)

print(first < second) # True

同樣的，== 操作符使用\_\_eq\_\_方法。

class Journey:

def \_\_init\_\_(self, location, destination, duration):

self.location = location

self.destination = destination

self.duration = duration

def \_\_eq\_\_(self, other):

return ((self.location == other.location) and

(self.destination == other.destination) and

(self.duration == other.duration))

first = Journey('Location A', 'Destination A', '30min')

second = Journey('Location B', 'Destination B', '30min')

print(first == second)

還有一些其他的定義。

\_\_sub\_\_() for -

\_\_mul\_\_() for \*

\_\_truediv\_\_() for /

\_\_ne\_\_() for !=

\_\_ge\_\_() for >=

\_\_gt\_\_() for >

**▍62、為類的物件定義自訂的可列印版本**

class Rectangle:

def \_\_init\_\_(self, a, b):

self.a = a

self.b = b

def \_\_repr\_\_(self):

return repr('Rectangle with area=' + str(self.a \* self.b))

print(Rectangle(3, 4)) # 'Rectangle with area=12'

**▍63、交換字串中字元的大小寫**

string = "This is just a sentence."

result = string.swapcase()

print(result) # tHIS IS JUST A SENTENCE.

**▍64、檢查字串是否都是空格**

string = " "

result = string.isspace()

print(result) # True

**▍65、檢查字串是否都是字母或數位**

name = "Password"

print(name.isalnum()) # True

name = "Secure Password "

print(name.isalnum()) # False

name = "S3cur3P4ssw0rd"

print(name.isalnum()) # True

name = "133"

print(name.isalnum()) # True

**▍66、檢查字串是否都是字母**

string = "Name"

print(string.isalpha()) # True

string = "Firstname Lastname"

print(string.isalpha()) # False

string = "P4ssw0rd"

print(string.isalpha()) # False

**▍67、根據參數刪除字元**

從右側開始。

string = "This is a sentence with "

print(string.rstrip()) # "This is a sentence with"

string = "this here is a sentence…..,,,,aaaaasd"

print(string.rstrip(".,dsa")) # "this here is a sentence"

同樣的，左側也能操作。

string = "ffffffffFirst"

print(string.lstrip("f")) # First

**▍68、檢查字串是否為數位**

string = "seven"

print(string.isdigit()) # False

string = "1337"

print(string.isdigit()) # True

string = "5a"

print(string.isdigit()) # False

string = "2\*\*5"

print(string.isdigit()) # False

**▍69、檢查字串是否為中文數位**

# 42673

string = "四二六七三"

print(string.isdigit()) # False

print(string.isnumeric()) # True

**▍70、檢查字串是否所有單詞都是大寫開頭**

string = "This is a sentence"

print(string.istitle()) # False

string = "10 Python Tips"

print(string.istitle()) # True

string = "How to Print A String in Python"

# False

print(string.istitle())

string = "PYTHON"

print(string.istitle()) # False

**▍71、在元組中使用負索引**

numbers = (1, 2, 3, 4)

print(numbers[-1]) # 4

print(numbers[-4]) # 1

**▍72、在元組中嵌套列表和元組**

mixed\_tuple = (("a"\*10, 3, 4), ['first', 'second', 'third'])

print(mixed\_tuple[1]) # ['first', 'second', 'third']

print(mixed\_tuple[0]) # ('aaaaaaaaaa', 3, 4)

**▍73、快速統計元素在清單中出現的次數**

names = ["Besim", "Albert", "Besim", "Fisnik", "Meriton"]

print(names.count("Besim")) # 2

**▍74、使用slice()獲取元素**

使用slice()獲取最後n個元素。

my\_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

slicing = slice(-4, None)

print(my\_list[slicing]) # [4, 5, 6]

print(my\_list[-3]) # 4

使用slice()做切片任務。

string = "Data Science"

slice\_object = slice(5, None)

print(string[slice\_object]) # Science

**▍75、計算元素在元組中出現的次數**

my\_tuple = ('a', 1, 'f', 'a', 5, 'a')

print(my\_tuple.count('a')) # 3

**▍76、獲取元組中元素的索引**

my\_tuple = ('a', 1, 'f', 'a', 5, 'a')

print(my\_tuple.index('f')) # 2

**▍77、步進獲得元組**

my\_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

print(my\_tuple[::3]) # (1, 4, 7, 10)

**▍78、通過索引獲取子元組**

my\_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

print(my\_tuple[3:]) # (4, 5, 6, 7, 8, 9, 10)

**▍79、將清單、集合、字典中所有元素刪除**

my\_list = [1, 2, 3, 4]

my\_list.clear()

print(my\_list) # []

my\_set = {1, 2, 3}

my\_set.clear()

print(my\_set) # set()

my\_dict = {"a": 1, "b": 2}

my\_dict.clear()

print(my\_dict) # {}

**▍80、合併集合**

使用union()方法，返回一個新集合。

first\_set = {4, 5, 6}

second\_set = {1, 2, 3}

print(first\_set.union(second\_set)) # {1, 2, 3, 4, 5, 6}

還可以使用update()方法，將第二個集合的元素插入到第一個集合中去。

first\_set = {4, 5, 6}

second\_set = {1, 2, 3}

first\_set.update(second\_set)

print(first\_set) # {1, 2, 3, 4, 5, 6}

**▍81、在函數裡輸出結果**

def is\_positive(number):

print("Positive" if number > 0 else "Negative") # Positive

is\_positive(-3)

**▍82、if語句中的多個條件**

math\_points = 51

biology\_points = 78

physics\_points = 56

history\_points = 72

my\_conditions = [math\_points > 50, biology\_points > 50,

physics\_points > 50, history\_points > 50]

if all(my\_conditions):

print("Congratulations! You have passed all of the exams.")

else:

print("I am sorry, but it seems that you have to repeat at least one exam.")

# Congratulations! You have passed all of the exams.

**▍83、在一個if語句中，至少滿足多個條件中的一個**

math\_points = 40

biology\_points = 78

physics\_points = 56

history\_points = 72

my\_conditions = [math\_points > 50, biology\_points > 50,

physics\_points > 50, history\_points > 50]

if any(my\_conditions):

print("Congratulations! You have passed all of the exams.")

else:

print("I am sorry, but it seems that you have to repeat at least one exam.")

# Congratulations! You have passed all of the exams.

**▍84、任何非空字串都為True**

print(bool("Non empty")) # True

print(bool("")) # False

**▍85、任何非空清單、元組、字典都為True**

print(bool([])) # False

print(bool(set([]))) # False

print(bool({})) # False

print(bool({"a": 1})) # True

**▍86、None、False、0都為False**

print(bool(False)) # False

print(bool(None)) # False

print(bool(0)) # False

**▍87、在函數中使用全域變數**

在函數無法直接修改全域變數的值。

string = "string"

def do\_nothing():

string = "inside a method"

do\_nothing()

print(string) # string

可通過修飾符global，修改全域變數的值。

string = "string"

def do\_nothing():

global string

string = "inside a method"

do\_nothing()

print(string) # inside a method

**▍88、計算字串或清單中元素的數量**

使用collections中的Counter計算字串或清單中元素的數量。

from collections import Counter

result = Counter("Banana")

print(result) # Counter({'a': 3, 'n': 2, 'B': 1})

result = Counter([1, 2, 1, 3, 1, 4, 1, 5, 1, 6])

print(result) # Counter({1: 5, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1})

**▍89、檢查2個字串是否為相同**

可以使用Counter()方法。

from collections import Counter

def check\_if\_anagram(first\_string, second\_string):

first\_string = first\_string.lower()

second\_string = second\_string.lower()

return Counter(first\_string) == Counter(second\_string)

print(check\_if\_anagram('testinG', 'Testing')) # True

print(check\_if\_anagram('Here', 'Rehe')) # True

print(check\_if\_anagram('Know', 'Now')) # False

可以使用sorted()方法。

def check\_if\_anagram(first\_word, second\_word):

first\_word = first\_word.lower()

second\_word = second\_word.lower()

return sorted(first\_word) == sorted(second\_word)

print(check\_if\_anagram("testinG", "Testing")) # True

print(check\_if\_anagram("Here", "Rehe")) # True

print(check\_if\_anagram("Know", "Now")) # False

**▍90、使用itertools中的count計算元素的數量**

from itertools import count

my\_vowels = ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']

current\_counter = count()

string = "This is just a sentence."

for i in string:

if i in my\_vowels:

print(f"Current vowel: {i}")

print(f"Number of vowels found so far: {next(current\_counter)}")

輸出如下。

Current vowel: i

Number of vowels found so far: 0

Current vowel: i

Number of vowels found so far: 1

Current vowel: u

Number of vowels found so far: 2

Current vowel: a

Number of vowels found so far: 3

Current vowel: e

Number of vowels found so far: 4

Current vowel: e

Number of vowels found so far: 5

Current vowel: e

Number of vowels found so far: 6

**▍91、對字串或清單的元素進行次數排序**

collections模組的Counter()，預設情況下是不會根據元素的頻率對它們進行排序的。

from collections import Counter

result = Counter([1, 2, 3, 2, 2, 2, 2])

print(result) # Counter({2: 5, 1: 1, 3: 1})

print(result.most\_common()) # [(2, 5), (1, 1), (3, 1)]

map()函數將給定函數應用於可反覆運算物件(清單、元組等)，然後返回結果(map對象)。

**▍92、查找清單中出現頻率最高的元素**

my\_list = ['1', 1, 0, 'a', 'b', 2, 'a', 'c', 'a']

print(max(set(my\_list), key=my\_list.count)) # a

**▍93、copy()和deepcopy()的區別**

淺拷貝: 拷貝父物件，但是不會拷貝物件的內部的子物件。

深拷貝: 拷貝父物件. 以及其內部的子物件。

下面是一個copy()的例子。

first\_list = [[1, 2, 3], ['a', 'b', 'c']]

second\_list = first\_list.copy()

first\_list[0][2] = 831

print(first\_list) # [[1, 2, 831], ['a', 'b', 'c']]

print(second\_list) # [[1, 2, 831], ['a', 'b', 'c']]

這裡是一個deepcopy()的例子。

import copy

first\_list = [[1, 2, 3], ['a', 'b', 'c']]

second\_list = copy.deepcopy(first\_list)

first\_list[0][2] = 831

print(first\_list) # [[1, 2, 831], ['a', 'b', 'c']]

print(second\_list) # [[1, 2, 3], ['a', 'b', 'c']]

**▍94、訪問字典中不存在的鍵時，避免報錯**

如果你想訪問字典一個不存在的鍵，代碼會報錯。

my\_dictonary = {"name": "Name", "surname": "Surname"}

print(my\_dictonary["age"])

錯誤如下。

KeyError: 'age'

可以通過使用defaultdict()，代碼將不會報錯。

from collections import defaultdict

my\_dictonary = defaultdict(str)

my\_dictonary['name'] = "Name"

my\_dictonary['surname'] = "Surname"

print(my\_dictonary["age"])

**▍95、構建反覆運算器**

class OddNumbers:

def \_\_iter\_\_(self):

self.a = 1

return self

def \_\_next\_\_(self):

x = self.a

self.a += 2

return x

odd\_numbers\_object = OddNumbers()

iterator = iter(odd\_numbers\_object)

print(next(iterator)) # 1

print(next(iterator)) # 3

print(next(iterator)) # 5

**▍96、刪除列表的重複項**

my\_set = set([1, 2, 1, 2, 3, 4, 5])

print(list(my\_set)) # [1, 2, 3, 4, 5]

**▍97、列印模組的安裝位置**

import pandas

print(pandas) # <module 'torch' from '/Users/...'

**▍98、使用not in檢查一個值是否在列表中**

odd\_numbers = [1, 3, 5, 7, 9]

even\_numbers = []

for i in range(9):

if i not in odd\_numbers:

even\_numbers.append(i)

print(even\_numbers) # [0, 2, 4, 6, 8]

**▍99、sort()和sorted()的區別**

sort()：對原始列表進行排序

sorted()：返回一個新的排序列表

groceries = ['milk', 'bread', 'tea']

new\_groceries = sorted(groceries)

# new\_groceries = ['bread', 'milk', 'tea']

print(new\_groceries)

# groceries = ['milk', 'bread', 'tea']

print(groceries)

groceries.sort()

# groceries = ['bread', 'milk', 'tea']

print(groceries)

**▍100、使用uuid模組生成唯一ID**

UUID代表唯一識別碼。

import uuid

# 根據主機ID、序號和當前時間生成UUID

print(uuid.uuid1()) # 308490b6-afe4-11eb-95f7-0c4de9a0c5af

# 生成一個隨機UUID

print(uuid.uuid4()) # 93bc700b-253e-4081-a358-24b60591076a

## 1. 易混淆操作

本節對一些 Python 易混淆的操作進行對比。

**1.1 有放回隨機採樣和無放回隨機採樣**

**import** random

random**.**choices(seq, k**=**1) *# 長度為k的list，有放回採樣*

random**.**sample(seq, k) *# 長度為k的list，無放回採樣*

**1.2 lambda 函數的參數**

func **=** **lambda** y: x **+** y *# x的值在函數運行時被綁定*

func **=** **lambda** y, x**=**x: x **+** y *# x的值在函式定義時被綁定*

**1.3 copy 和 deepcopy**

**import** copy

y **=** copy**.**copy(x) *# 只複製最頂層*

y **=** copy**.**deepcopy(x) *# 複製所有嵌套部分*

複製和變數別名結合在一起時，容易混淆：

a **=** [1, 2, [3, 4]]

*# Alias.*

b\_alias **=** a

**assert** b\_alias **==** a **and** b\_alias **is** a

*# Shallow copy.*

b\_shallow\_copy **=** a[:]

**assert** b\_shallow\_copy **==** a **and** b\_shallow\_copy **is** **not** a **and** b\_shallow\_copy[2] **is** a[2]

*# Deep copy.*

**import** copy

b\_deep\_copy **=** copy**.**deepcopy(a)

**assert** b\_deep\_copy **==** a **and** b\_deep\_copy **is** **not** a **and** b\_deep\_copy[2] **is** **not** a[2]

對別名的修改會影響原變數，（淺）複製中的元素是原清單中元素的別名，而深層複製是遞迴的進行複製，對深層複製的修改不影響原變數。

**1.4 == 和 is**

x **==** y *# 兩引用物件是否有相同值*

x **is** y *# 兩引用是否指向同一物件*

**1.5 判斷類型**

type(a) **==** int *# 忽略物件導向設計中的多態特徵*

isinstance(a, int) *# 考慮了物件導向設計中的多態特徵*

**1.6 字串搜索**

str**.**find(sub, start**=None**, end**=None**); str**.**rfind(**...**) *# 如果找不到返回-1*

str**.**index(sub, start**=None**, end**=None**); str**.**rindex(**...**) *# 如果找不到拋出ValueError異常*

**1.7 List 後向索引**

這個只是習慣問題，前向索引時下標從0開始，如果反向索引也想從0開始可以使用~。

print(a[**-**1], a[**-**2], a[**-**3])

print(a[**~**0], a[**~**1], a[**~**2])

## 2. C/C++ 使用者使用指南

不少 Python 的用戶是從以前 C/C++ 遷移過來的，這兩種語言在語法、代碼風格等方面有些不同，本節簡要進行介紹。

**2.1 很大的數和很小的數**

C/C++ 的習慣是定義一個很大的數字，Python 中有 inf 和 -inf：

a **=** float('inf')

b **=** float('-inf')

**2.2 布林值**

C/C++ 的習慣是使用 0 和非 0 值表示 True 和 False， Python 建議直接使用 True 和 False 表示布林值。

a **=** **True**

b **=** **False**

**2.3 判斷為空**

C/C++ 對空指針判斷的習慣是 if (a) 和 if (!a)。Python 對於 None 的判斷是：

**if** x **is** **None**:

**pass**

如果使用 if not x，則會將其他的物件（比如長度為 0 的字串、清單、元組、字典等）都會被當做 False。

**2.4 交換值**

C/C++ 的習慣是定義一個臨時變數，用來交換值。利用 Python 的 Tuple 操作，可以一步到位。

a, b **=** b, a

**2.5 比較**

C/C++ 的習慣是用兩個條件。利用 Python 可以一步到位。

**if** 0 **<** a **<** 5:

**pass**

**2.6 類成員的 Set 和 Get**

C/C++ 的習慣是把類成員設為 private，通過一系列的 Set 和 Get 函數存取其中的值。在 Python 中雖然也可以通過 @property、@setter、@deleter 設置對應的 Set 和 Get 函數，我們應避免不必要的抽象，這會比直接訪問慢 4 - 5 倍。

**2.7 函數的輸入輸出參數**

C/C++ 的習慣是把輸入輸出參數都列為函數的參數，通過指標改變輸出參數的值，函數的返回值是執行狀態，函式呼叫方對返回值進行檢查，判斷是否成功執行。在 Python 中，不需要函式呼叫方進行返回值檢查，函數中遇到特殊情況，直接拋出一個異常。

**2.8 讀文件**

相比 C/C++，Python 讀檔要簡單很多，打開後的檔是一個可反覆運算物件，每次返回一行內容。

**with** open(file\_path, 'rt', encoding**=**'utf-8') **as** f:

**for** line **in** f:

print(line) *# 末尾的\n會保留*

**2.9 檔路徑拼接**

C/C++ 的習慣通常直接用 + 將路徑拼接，這很容易出錯，Python 中的 os.path.join 會自動根據作業系統不同補充路徑之間的 / 或 \ 分隔符號號：

**import** os

os**.**path**.**join('usr', 'lib', 'local')

**2.10 解析命令列選項**

雖然 Python 中也可以像 C/C++ 一樣使用 sys.argv 直接解析命令列選擇，但是使用 argparse 下的 ArgumentParser 工具更加方便，功能更加強大。

**2.11 調用外部命令**

雖然 Python 中也可以像 C/C++ 一樣使用 os.system 直接調用外部命令，但是使用 subprocess.check\_output 可以自由選擇是否執行 Shell，也可以獲得外部命令執行結果。

**import** subprocess

*# 如果外部命令返回值非0，則拋出subprocess.CalledProcessError異常*

result **=** subprocess**.**check\_output(['cmd', 'arg1', 'arg2'])**.**decode('utf-8')

*# 同時收集標準輸出和標準錯誤*

result **=** subprocess**.**check\_output(['cmd', 'arg1', 'arg2'], stderr**=**subprocess**.**STDOUT)**.**decode('utf-8')

*# 執行shell命令（管道、重定向等），可以使用shlex.quote()將參數雙引號引起來*

result **=** subprocess**.**check\_output('grep python | wc > out', shell**=True**)**.**decode('utf-8')

**2.12 不重複造輪子**

不要重複造輪子，Python稱為batteries included即是指Python提供了許多常見問題的解決方案。

## 3. 常用工具

**3.1 讀寫 CSV 文件**

**import** csv

*# 無header的讀寫*

**with** open(name, 'rt', encoding**=**'utf-8', newline**=**'') **as** f: *# newline=''讓Python不將換行統一處理*

**for** row **in** csv**.**reader(f):

print(row[0], row[1]) *# CSV讀到的資料都是str類型*

**with** open(name, mode**=**'wt') **as** f:

f\_csv **=** csv**.**writer(f)

f\_csv**.**writerow(['symbol', 'change'])

*# 有header的讀寫*

**with** open(name, mode**=**'rt', newline**=**'') **as** f:

**for** row **in** csv**.**DictReader(f):

print(row['symbol'], row['change'])

**with** open(name, mode**=**'wt') **as** f:

header **=** ['symbol', 'change']

f\_csv **=** csv**.**DictWriter(f, header)

f\_csv**.**writeheader()

f\_csv**.**writerow({'symbol': xx, 'change': xx})

注意，當 CSV 文件過大時會報錯：\_csv.Error: field larger than field limit (131072)，通過修改上限解決

**import** sys

csv**.**field\_size\_limit(sys**.**maxsize)

csv 還可以讀以 \t 分割的資料

f **=** csv**.**reader(f, delimiter**=**'\t')

**3.2 反覆運算器工具**

itertools 中定義了很多反覆運算器工具，例如子序列工具：

**import** itertools

itertools**.**islice(iterable, start**=None**, stop, step**=None**)

*# islice('ABCDEF', 2, None) -> C, D, E, F*

itertools**.**filterfalse(predicate, iterable) *# 過濾掉predicate為False的元素*

*# filterfalse(lambda x: x < 5, [1, 4, 6, 4, 1]) -> 6*

itertools**.**takewhile(predicate, iterable) *# 當predicate為False時停止反覆運算*

*# takewhile(lambda x: x < 5, [1, 4, 6, 4, 1]) -> 1, 4*

itertools**.**dropwhile(predicate, iterable) *# 當predicate為False時開始反覆運算*

*# dropwhile(lambda x: x < 5, [1, 4, 6, 4, 1]) -> 6, 4, 1*

itertools**.**compress(iterable, selectors) *# 根據selectors每個元素是True或False進行選擇*

*# compress('ABCDEF', [1, 0, 1, 0, 1, 1]) -> A, C, E, F*

序列排序：

sorted(iterable, key**=None**, reverse**=False**)

itertools**.**groupby(iterable, key**=None**) *# 按值分組，iterable需要先被排序*

*# groupby(sorted([1, 4, 6, 4, 1])) -> (1, iter1), (4, iter4), (6, iter6)*

itertools**.**permutations(iterable, r**=None**) *# 排列，返回值是Tuple*

*# permutations('ABCD', 2) -> AB, AC, AD, BA, BC, BD, CA, CB, CD, DA, DB, DC*

itertools**.**combinations(iterable, r**=None**) *# 組合，返回值是Tuple*

itertools**.**combinations\_with\_replacement(**...**)

*# combinations('ABCD', 2) -> AB, AC, AD, BC, BD, CD*

多個序列合併：

itertools**.**chain(**\***iterables) *# 多個序列直接拼接*

*# chain('ABC', 'DEF') -> A, B, C, D, E, F*

**import** heapq

heapq**.**merge(**\***iterables, key**=None**, reverse**=False**) *# 多個序列按順序拼接*

*# merge('ABF', 'CDE') -> A, B, C, D, E, F*

zip(**\***iterables) *# 當最短的序列耗盡時停止，結果只能被消耗一次*

itertools**.**zip\_longest(**\***iterables, fillvalue**=None**) *# 當最長的序列耗盡時停止，結果只能被消耗一次*

**3.3 計數器**

計數器可以統計一個可反覆運算物件中每個元素出現的次數。

**import** collections

*# 創建*

collections**.**Counter(iterable)

*# 頻次*

collections**.**Counter[key] *# key出現頻次*

*# 返回n個出現頻次最高的元素和其對應出現頻次，如果n為None，返回所有元素*

collections**.**Counter**.**most\_common(n**=None**)

*# 插入/更新*

collections**.**Counter**.**update(iterable)

counter1 **+** counter2; counter1 **-** counter2 *# counter加減*

*# 檢查兩個字串的組成元素是否相同*

collections**.**Counter(list1) **==** collections**.**Counter(list2)

**3.4 帶預設值的 Dict**

當訪問不存在的 Key 時，defaultdict 會將其設置為某個預設值。

**import** collections

collections**.**defaultdict(type) *# 當第一次訪問dict[key]時，會無參數調用type，給dict[key]提供一個初始值*

**3.5 有序 Dict**

**import** collections

collections**.**OrderedDict(items**=None**) *# 反覆運算時保留原始插入順序*

## 4. 高性能程式設計和調試

**4.1 輸出錯誤和警告資訊**

向標準錯誤輸出資訊

**import** sys

sys**.**stderr**.**write('')

輸出警告資訊

**import** warnings

warnings**.**warn(message, category**=UserWarning**)

*# category的取值有DeprecationWarning, SyntaxWarning, RuntimeWarning, ResourceWarning, FutureWarning*

控制警告消息的輸出

$ python **-**W all *# 輸出所有警告，等同於設置warnings.simplefilter('always')*

$ python **-**W ignore *# 忽略所有警告，等同於設置warnings.simplefilter('ignore')*

$ python **-**W error *# 將所有警告轉換為異常，等同於設置warnings.simplefilter('error')*

**4.2 代碼中測試**

有時為了調試，我們想在代碼中加一些代碼，通常是一些 print 語句，可以寫為：

*# 在代碼中的debug部分*

**if** \_\_debug\_\_:

**pass**

一旦調試結束，通過在命令列執行 -O 選項，會忽略這部分代碼：

$ python -0 main.py

**4.3 代碼風格檢查**

使用 pylint 可以進行不少的代碼風格和語法檢查，能在運行之前發現一些錯誤

pylint main.py

**4.4 代碼耗時**

耗時測試

$ python -m cProfile main.py

測試某代碼塊耗時

*# 代碼塊耗時定義*

from contextlib import contextmanager

from time import perf\_counter

@contextmanager

def timeblock**(**label**)**:

tic **=** perf\_counter**()**

try:

yield

finally:

toc **=** perf\_counter**()**

print**(**'%s : %s' % **(**label, toc - tic**))**

*# 代碼塊耗時測試*

with timeblock**(**'counting'**)**:

pass

代碼耗時優化的一些原則

* 專注於優化產生性能瓶頸的地方，而不是全部代碼。
* 避免使用全域變數。區域變數的查找比全域變數更快，將全域變數的代碼定義在函數中運行通常會快 15%-30%。
* 避免使用.訪問屬性。使用 from module import name 會更快，將頻繁訪問的類的成員變數 self.member 放入到一個區域變數中。
* 儘量使用內置資料結構。str, list, set, dict 等使用 C 實現，運行起來很快。
* 避免創建沒有必要的中間變數，和 copy.deepcopy()。
* 字串拼接，例如 a + ':' + b + ':' + c 會創造大量無用的中間變數，':',join([a, b, c]) 效率會高不少。另外需要考慮字串拼接是否必要，例如 print(':'.join([a, b, c])) 效率比 print(a, b, c, sep=':') 低。

## 5. Python 其他技巧

**5.1 argmin 和 argmax**

items **=** [2, 1, 3, 4]

argmin **=** min(range(len(items)), key**=**items**.**\_\_getitem\_\_)

argmax同理。

**5.2 轉置二維列表**

A = [['a11', 'a12'], ['a21', 'a22'], ['a31', 'a32']]

A\_transpose = list(zip(\*A)) # list of tuple

A\_transpose = list(list(col) for col in zip(\*A)) # list of list

**5.3 一維列表展開為二維列表**

A **=** [1, 2, 3, 4, 5, 6]

*# Preferred.*

list(zip(**\***[iter(A)] **\*** 2))

## 列表碾平式

需求： 將[[1,2],[3,4]] 轉換為[1,2,3,4]，具體實現有以下幾種方法

test\_list = [[1,2],[3,4]]

1. from itertools import chain

list(chain.from\_iterable(test\_list))

結果：[1, 2, 3, 4]

2. from itertools import chain

list(chain(\*test\_list))

結果：[1, 2, 3, 4]

3. sum(test\_list, [])

結果：[1, 2, 3, 4]

4. [x for y in test\_list for x in y]

結果：[1, 2, 3, 4]

5. 萬能方法（遞迴）

func = lambda x: [y for t in x for y in func(t)] if type(x) is list else [x]

func(test\_list)

結果：[1, 2, 3, 4]

PS： 專案中，難免會有類似的需求，對於結構嵌套一致的情況，上述的1，2，3，4方法都可以很好的解決(不建議用for迴圈嵌套的方式，那是最low的方法，沒有之一)；對於結構嵌套不一致的情況，第5種方法就派上了用場，其採用了遞迴的思想，堪稱萬能的方法，屢試不爽。專案中，大家可以根據實際應用場景來挑選最適合自己的方法。肚中有糧，心中不慌；大家可以把上述方法都記下來，以做到有的放矢。

## 三目操作符

對於Python的三元運算式，想必大家對if else都不會感到陌生，但是對and or操作想必是另一種感覺了，不過大家對其他語言的? :應該不會陌生，沒錯，這次的主角and or就和? :有著異曲同工之妙。

代碼參考：

1 == 1 and 2 or 3 返回2

1 == 2 and 2 or 3 返回3

PS：習慣於if else的同學偶爾用下and or是不是會給人耳目一新的感覺。

## and or 分開來用

* [x] and 用法如下：

1 and 2 and 3 返回3

1 and 2 and '' 返回''

'' and 2 and 0 返回''

PS：如果都為真則返回最後一個值，如果其中某些值為假，則返回第一個為假的值

* [x] or 用法如下

1 or '' or 0 返回1

'' or 0 or [] 返回[]

PS：如果都為假返回最後一個值，如果其中某些值為真，則返回第一個為真的值

使用場景：在項目中我們經常會有這樣的需求，在將一個字典更新之後還想要返回更新後的字典，這是我們就可以這樣寫：dic = dic1.update(dic2) or dic1

## 列表推導式

需求：將[1,2,3]中的每一項都加1

good: [x+1 for x in [1,2,3]]

bad: def add\_list(goal\_list):

tmp\_list = []

for x in goal\_list:

tmp\_list.append(x+1)

PS: 清單推導式底層是用C實現的，其執行速度要比for迴圈快好多

## vars() 用法

代碼參考：

```

def func(a, b, c):

print vars()

執行func(1, 2, 3)

輸出：{"a":1,"b":2,"c":3}

PS: vars()的值為字典，其鍵值對來源於當前作用域的所有變數。

```

使用場景：在調用他人介面或方法時，需要將傳入的參數列印以記錄日誌，此刻vars()便派上用場了。

## 偏函數之partial

代碼示例：

import functools

def add(a, b):

return a + b

add(4, 2)

6

plus3 = functools.partial(add, 3)

plus5 = functools.partial(add, 5)

plus3(4)

7

plus3(7)

10

plus5(10)

15

實際使用心得：

之前做物件存儲的專案中，我需要同時去調用三個一樣的介面(大部分參數一樣)下面是我的部分代碼，供大家參考

my\_thread = functools.partial(myThread, dic, zone\_id, start, end, customer\_id)

my\_thread1 = my\_thread("day", "gets", "2")

my\_thread1 = my\_thread("day", "original\_traffic", "3")

使用場景：  
當我們需要同時去調用一個函數，並且發現大部分參數一致的時候，便可以採取上述方法，一來代碼簡潔，二來可讀性高。

## Python搭建簡易服務

* Python搭建簡易郵件伺服器：python -m smtpd -n -c DebuggingServer localhost:1025
* Python搭建簡易web伺服器：
  + Python2： python -m SimpleHTTPServer port
  + Python3： python -m http.server port
* Python搭建簡易ftp服務

pip install pyftpdlib

python -m pyftpdlib -p 21

ftp://localhost:21

## 遍歷字典

在專案中遍歷字典是很常見的需求，下面介紹下常用的方法並做下比較：

dic = {'name': 'peter', 'age': 27}

1. for key, value in dic.items():

print key, value

2. for key, value in dic.iteritems():

print key, value

PS:  
iteritems和items的區別在於iteritems採用了生成器的原理，只有在需要的時候才會把值生成，其之間的區別類似於range和 xrange；readline和 xreadline

## 記憶體管理

Python的記憶體管理主要分為引用計數和垃圾回收機制兩大部分，且看下麵代碼：

* [ ] 記憶體分配：

a = 1

b = 1

a is b True

---------------------

a = 1000

b = 1000

a is b False

PS: 在Python中，整數和短小的字元，Python都會緩存這些物件，以便重複使用。當我們創建多個等於1的引用時，實際上是讓這些引用指向了同一個物件。

* [ ] 引用計數：

在Python中，所謂引用計數(reference count)是指所有指向該物件的引用的總數;

我們可以使用sys包中的getrefcount(),來查看某個物件的引用計數。需要注意的是，當使用該函數查看某個物件的引用計數時，實際上是臨時創建了該物件的一個新的引用，所有使用getrefcount()所得到的結果，會比期望的值多1。

from sys import getrefcount

aa = 'test refcount' print(getrefcount(a))

bb = aa

print(getrefcount(a))

PS: 由於上述原因，兩個getrefcount()將返回2和3，並不是期望的1和2.

* [ ] 引用減少

引用減少大致分為兩類：

* + 指向該物件的引用指向了其他物件

from sys import getrefcount

aa = 'test refcount'

bb = aa

print(getrefcount(aa)) 3

bb = 1

print(getrefcount(aa)) 2

* + 使用del關鍵字顯示的刪除某個引用

from sys import getrefcount

aa = 'test refcount'

bb = aa

print(getrefcount(aa)) 3

del bb

print(getrefcount(aa)) 2

* [ ] 垃圾回收

不斷的創建物件，如果不及時銷毀的話，那Python的體積會越來越大，再大的記憶體也會有耗完的時候；不用像C語言那樣，需要手動的去管理記憶體、Python已經幫我們做好了(Python的垃圾回收機制)，你只需要去關心你的業務邏輯即可，其他的都交給Python來處理。

從原理上講，當Python中某個對象的引用計數降為0時，該物件就應該被回收。但是頻繁的啟動垃圾回收機制畢竟是個很耗時的問題；因此Python只有在特定條件下(當Python中被分配物件和取消分配物件的次數之間的差值達到某個閾值時)，Python會自動啟動垃圾回收機制。

我們可以通過gc模組的get\_threshold()方法，查看該閾值：

import gc

print(gc.get\_threshold())

該方法會返回(700, 10, 10)，後面的倆10是與分代回收相關的，稍後講解。700便是垃圾回收機制啟動的閾值。可以通過gc模組中的set\_threshold()方法重新設定該值。

當然了，我們也可以手動啟動垃圾回收機制，使用gc.collect()即可。

* [ ] 分代回收

Python同時採用了分代回收的機制，設想一下：存活越久的物件、越不可能是垃圾物件。程式在運行時，往往會產生大量的臨時物件，程式結束之後，這些臨時物件的生命週期也就隨之告一段落。但有一些對象會被長期佔用，垃圾回收機制在啟動的時候會減少掃描到他們的頻率。

Python將所有物件分為0，1，2三代。所有新創建的物件都是0代，當垃圾回收機制在啟動多次0代機制並掃描到他們的時候，這些物件如果依然存活在記憶體的話，他們就會被歸入下一代對象，以此類推。

剛才上面所提到的(700, 10, 10)三個參數後面的倆10所代表的意思是：每10次0代垃圾回收，會配合1次1代的垃圾回收；而每10次1代的垃圾回收會配合1次的2代垃圾回收。

當然我們同樣可以使用set\_threshold()來調整此策略的比例，比如對1代物件進行更頻繁的掃描。

import gc

gc.set\_threshold(700, 5, 10)

## 新式類、經典類

* [ ] 新式類: 顯示的繼承了object的類

class A(object):

attr = 1

class B(A):

pass

class C(A):

attr = 3

class D(B, C):

pass

if \_\_name\_\_ == '\_\_main\_\_':

d = D()

print 'attr = ', d.attr # attr = 3

* [ ] 經典類：沒有繼承自object的類

class A():

attr = 1

class B(A):

pass

class C(A):

attr = 3

class D(B, C):

pass

if \_\_name\_\_ == '\_\_main\_\_':

d = D()

print 'attr = ', d.attr # attr = 1

PS: 通過以上代碼的輸出結果可以看出，新式類會廣度搜索，也就是一層層的向上搜索；經典類是深度優先，即遇到一個超類點就向上搜索。

## 裝飾器

Python的裝飾器被稱為Python的語法糖，哪裡需要粘哪裡。

代碼示例：

@makeh1

@makeeitalic

def say():

return 'Peter'

我們希望輸出結果為：<h1><i>Peter</i></h1>

去看看[官方文檔](https://link.zhihu.com/?target=https%3A//link.juejin.im/%3Ftarget%3Dhttps%253A%252F%252Fdocs.python.org%252F2%252Freference%252Fcompound_stmts.html%2523function)，答案就看下面：

def makeh1(func):

def wrp():

return "<h1>" + func() + "</h1>"

return wrp

def makeeitalic(func):

def wrp():

return "<i>" + func() + "</i>"

return wrp

@makeh1

@makeeitalic

def say():

return 'Hello Peter'

print say()

輸出：<h1><i>Hello Peter</i></h1>

實際應用場景：

使用過django的小夥伴想必都用過login\_required裝飾器，但是如果用戶沒登錄的話login\_required會重定向到登錄頁面；在做web開發的過程中，我們會經常用  
ajax非同步提交資料到後臺，這時如果再繼續使用原有的login\_required裝飾器肯定是不行了(該裝飾器不會重定向到登錄頁面，ajax也沒有任何返回結果)，下麵我們改變下原有代碼：

from django.shortcuts import HttpResponse

import json

def is\_authenticat(func):

def wrp(req, \*\*kwargs):

if req.user.is\_authenticated():

return func(req, \*\*kwargs)

else:

json\_str = {'status': 0, 'msg': u'請登錄'}

return HttpResponse(json.dumps(json\_str), content\_type='application/json')

return wrp

上述代碼便很好的解決了問題，也算是對Python裝飾器的一個很好的使用場景。

## all or any

Python 語言如此流行的眾多原因之一，是因為它具有很好的可讀性和表現力。

人們經常開玩笑說 Python 是可執行的偽代碼。當你可以像這樣寫代碼時，就很難反駁。

x = [True, True, False]

if any(x):

print("至少有一個True")

if all(x):

print("全是True")

if any(x) and not all(x):

print("至少一個True和一個False")

複製代碼

## bashplotlib

你有沒有想過在控制台中繪製圖形嗎？

Bashplotlib 是一個 Python 庫，他能夠幫助我們在命令列(粗曠的環境)中繪製資料。

# 模組安裝

pip install bashplotlib

# 繪製實例

import numpy as np

from bashplotlib.histpgram import plot\_hist

arr = np.ramdom.normal(size=1000, loc=0, scale=1)

plot\_hist(arr, bincount=50)

複製代碼

## collections

Python 有一些很棒的默認資料類型，但有時它們的行為並不完全符合你的期望。

幸運的是，Python 標準庫提供了 collections 模組[1]。這個方便的附加元件為你提供了更多的資料類型。

from collections import OrderedDict, Counter

# 記住鍵的添加順序！

x = OrderedDict(a=1, b=2, c=3)

# 統計每個字元出現的頻率

y = Counter("Hello World!")

複製代碼

## dir

有沒有想過如何查看 Python 物件內部，查看它具有哪些屬性？在命令列中輸入：

dir()

dir("Hello World")

dir(dir)

複製代碼

## emoji

emoji[3] 是日本在無線通訊中所使用的視覺情感符號，繪指圖畫，文字指的則是字元，可用來代表多種表情，如笑臉表示笑、蛋糕表示食物等。在中國大陸，emoji通常叫做“小黃臉”，或者直稱emoji。

# 安裝模組

pip install emoji

# 做個嘗試

from emoji import emojize

print(emojize(":thumbs\_up:"))

複製代碼

## fromfuture import

Python 流行的結果之一，總是有新版本正在開發中。新版本意味著新功能 —— 除非你的版本已過時。

不過不要擔心。使用該\_\_future\_\_模組[4]可以説明你用Python的未來版本導入功能。從字面上看，這就像是時間旅行、魔法或其他東西

from \_\_future\_\_ import print\_function

print("Hello World!")

複製代碼

## geogy

地理，對大多數程式師來說是一個具有挑戰性的領域。在獲取地理資訊或者繪製地圖時，也會遇到不少問題。這個geopy 模組[5]讓地理相關內容變得非常容易。

pip install geopy

複製代碼

它通過抽象一系列不同地理編碼服務的 API 來工作。通過它，你能夠獲得一個地方的完整街道位址、緯度、經度甚至海拔高度。

還有一個有用的距離類。它以最好的測量單位計算兩個位置之間的距離。

from geopy import GoogleV3

place = "221b Baker Street, London"

location = GoogleV3().geocode(place)

print(location.address)

print(location.location)

複製代碼

## howdoi

當你使用terminal終端程式設計時，通過在遇到問題後會在StackOverflow上搜索答案，完後會回到終端繼續程式設計，此時有時會不記得你之前查到的解決方案，此時需要重新查看StackOverflow，但又不想離開終端，那麼此時你需要用到這個有用的命令列工具howdoi[6]。

pip install howdoi

複製代碼

無論你有什麼問題，都可以問它，它會盡力回復。

howdoi vertical align css

howdoi for loop in java

howdoi undo commits in git

複製代碼

但請注意——它會在 StackOverflow 的最佳答案中抓取代碼。它可能並不總是提供最有用的資訊......

howdoi exit vim

**1、如何使用 print 輸出日誌**

初學者喜歡使用 print 來調試代碼，並記錄程式運行過程。

但是 print 只會將內容輸出到終端上，不能持久化到日誌檔中，並不利於問題的排查。

如果你熱衷於使用 print 來調試代碼（雖然這並不是最佳做法），記錄程式運行過程，那麼下面介紹的這個 print 用法，可能會對你有用。

Python 3 中的 print 作為一個函數，由於可以接收更多的參數，所以功能變為更加強大，指定一些參數可以將 print 的內容輸出到日誌檔中。

>>> with open('test.log', mode='w') as f:

... print('hello, python', file=f, flush=True)

>>> exit()

$ cat test.log

hello, python

**2、規則運算式基本配方**

import re

pattern = re.compile(r”\d\d”)

print(re.search(pattern,"Let's find the number 23").group())

# or

print(re.findall(pattern, “Let's find the number 23”))[0]

# Outputs

'23'

'23'

Regex 對於許多 python 管道來說都是必須的，所以記住***核心Regex方法***很有用處。

**3、將嵌套 for 迴圈寫成單行**

我們經常會如下這種嵌套的 for 迴圈代碼

list1 = range(1,3)

list2 = range(4,6)

list3 = range(7,9)

for item1 in list1:

for item2 in list2:

for item3 in list3:

print(item1+item2+item3)

這裡僅僅是三個 for 迴圈，在實際編碼中，有可能會有更層。這樣的代碼，可讀性非常的差，很多人不想這麼寫，可又沒有更好的寫法。

這裡介紹一種常用的寫法，***使用 itertools 這個庫***來實現更優雅易讀的代碼。

from itertools import product

list1 = range(1,3)

list2 = range(4,6)

list3 = range(7,9)

for item1,item2,item3 in product(list1, list2, list3):

print(item1+item2+item3)

輸出：

$ python demo.py

12

13

13

14

13

14

14

15

**4、如何在運行狀態查看原始程式碼？**

查看函數的原始程式碼，我們通常會使用 IDE 來完成。比如在 PyCharm 中，你可以 Ctrl + 滑鼠點擊 進入函數的原始程式碼。

那如果沒有 IDE 呢？想使用一個函數時，如何知道這個函數需要接收哪些參數呢？

這時可以使用***inspect 來代替 IDE***幫助你完成這些事

# demo.py

import inspect

def add(x, y):

return x + y

print("===================")

print(inspect.getsource(add))

運行結果：

$ python demo.py

===================

def add(x, y):

return x + y

這裡我想先說一個點：不管你是通過各種管道學Python還是自學？還是說你大學在學習，甚至於說有別的程式設計的基礎，一定要注重一個東西：***完整的知識體系***。對於每一個自學的人，按照這個體系去打好基礎，你未來的路會走得更穩重。

適用人群：零基礎 / 基礎不紮實者，學Python都從這裡開始

[零基礎學Python有什麼建議？687 贊同 · 7 評論回答](https://www.zhihu.com/question/451604793/answer/1850553245)

**5、最快查看包搜索路徑的方式**

當你使用 import 導入一個包或模組時，Python 會去一些目錄下查找，而這些目錄是有優先順序順序的，正常人會使用 sys.path 查看。

>>> import sys

>>> from pprint import pprint

>>> pprint(sys.path)

['',

'/usr/local/Python3.7/lib/python37.zip',

'/usr/local/Python3.7/lib/python3.7',

'/usr/local/Python3.7/lib/python3.7/lib-dynload',

'/home/wangbm/.local/lib/python3.7/site-packages',

'/usr/local/Python3.7/lib/python3.7/site-packages']

>>>

有沒有更快的方式呢？一行命令即可解決！

[wangbm@localhost ~]$ python3 -m site

sys.path = [

'/home/wangbm',

'/usr/local/Python3.7/lib/python37.zip',

'/usr/local/Python3.7/lib/python3.7',

'/usr/local/Python3.7/lib/python3.7/lib-dynload',

'/home/wangbm/.local/lib/python3.7/site-packages',

'/usr/local/Python3.7/lib/python3.7/site-packages',

]

USER\_BASE: '/home/wangbm/.local' (exists)

USER\_SITE: '/home/wangbm/.local/lib/python3.7/site-packages' (exists)

ENABLE\_USER\_SITE: True

從輸出你可以發現，這個列的路徑會比 sys.path 更全，它包含了使用者環境的目錄。

**6、如何快速計算函數執行時間**

計算一個函數的執行時間，有人可能會這樣寫：

import time

start = time.time()

# run the function

end = time.time()

print(end-start)

你看看你為了計算函數執行時間，寫了幾行代碼了？

有沒有可以更方便計算的呢？

***內置模組叫 timeit！***使用它，只用一行代碼即可

import time

import timeit

def run\_sleep(second):

print(second)

time.sleep(second)

# 只用這一行

print(timeit.timeit(lambda :run\_sleep(2), number=5))

運行結果

2

2

2

2

2

10.020059824

**7、partial 函數**

from functools import partial

def multiply(x,y):

return x\*y

dbl = partial(multiply,2)

print(dbl)

print(dbl(4))

# Outputs

functools.partial(<function multiply at 0x7f16be9941f0>, 2)

8

在這裡，我們創建一個函數，它複製另一個函數，但使用的參數比原始函數少，這樣就可以使用它將該參數應用於多個不同的參數。

**8、使用 hasattr() 內置方法獲取object屬性**

class SomeClass:

def \_\_init\_\_(self):

self.attr1 = 10

def attrfunction(self):

print("Attreibute")

hasattr(SomeClass, "attrfunction")

# Output

True

**9、使用 defaultdict 和 lambda 函數創建字典**

from collections import defaultdict

import numpy as np

q = defaultdict(lambda: np.zeros(5))

# Example output

In : q[0]

Out: array([0., 0., 0., 0., 0.])

defaultdicts不會引發KeyError，任何不存在的鍵都會獲取預設工廠返回的值。  
在上述代碼，預設工廠是一個lambda函數，它為給定的任何鍵返回一個預設NumPy陣列，其中包含5個零。

**10、使用集合從兩個清單中獲得差異**

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

list2 = [3,4,5]

print(list(set(list1) — set(list2)))

# or

print(set(lista1).difference(set(lista2)))

# Outputs

[1,2]

{1,2}

在這裡，集合有助於獲得兩個python列表之間的差異，這兩個列表既是一個清單，也是一個集合。

**11、使用isinstance（）檢查變數是否為給定類型**

isinstance(1, int)

#Output

True

**12、使用map（）列印清單中的數位**

一種比迴圈列印清單內容更快更有效的方法

list1 = [1,2,3]

list(map(print, list1))

# Output

1

2

3

**13、使用.join（）方法格式化datetime日期**

from datetime import datetime

date = datetime.now()

print("-".join([str(date.year), str(date.month), str(date.day)])

**14、將兩個具有相同規則的清單隨機化**

import numpy as np

x = np.arange(100)

y = np.arange(100,200,1)

idx = np.random.choice(np.arange(len(x)), 5, replace=False)

x\_sample = x[idx]

y\_sample = y[idx]

print(x\_sample)

print(y\_sample)

# Outputs

array([68, 87, 41, 16, 0])

array([168, 187, 141, 116, 100])

**15、對輸入的字串“消毒”**

對使用者輸入的內容“消毒”，這問題幾乎適用於你編寫的所有程式。

通常將字元轉換為小寫或大寫就足夠了，有時還可以使用規則運算式來完成工作，但是對於複雜的情況，還有更好的方法：

user\_input = "This\nstring has\tsome whitespaces...\r\n"

character\_map = {

ord('\n') : ' ',

ord('\t') : ' ',

ord('\r') : None

}

user\_input.translate(character\_map) # This string has some whitespaces... "

在上述代碼，可以看到空格字元“ \n”和“ \t”被單個空格替換了，而“ \r”則被完全刪除。

這是一個簡單的示例，但是我們可以更進一步，使用unicodedata 庫及其 combining() 函數，來生成更大的重映射表（remapping table），並用它來刪除字串中所有的重音。

**16、反轉字串**

編寫一些代碼來反轉字串

def reverse\_string(string):

result=""

for c in range(len(string),-1,-1):

result = result + string[c]

return result

看起來有點亂吧， 用另一種方式來表達：

def reverse\_string(string):

result = [ string[c] for c in range(len(string),-1,-1)]

return "".join(result)

看起來不錯吧，其實使用切片的方法可以更好，方法如下：

def reverse\_string(string):

return string[::-1]

**17、將數位與數位進行求和**

將數位與數位進行求和，這是一個非常簡單的問題，我們可以用傳統方法解決這個問題：

def sum\_a\_num(num):

sum = 0

while num > 0:

sum+= num%10

num//=10

return sum

這是可以的，但在緊張的情況下，最終可能會與運運算元發生拼寫錯誤或錯誤，最終得到錯誤的結果，並花費數小時嘗試調試代碼。

想要避免這種情況，有一個更好的方法做到這一點：

def sum\_a\_num(num):

return sum(list(map(int,str(num))))

上述代碼所做的是：

* 將 num 轉換為字串；
* map() 函數在字串上遍數，並將每個字元轉換為整數；
* list() 函數將映射物件轉換為清單，然後求和；

num = 2367

str(num) = '2367'

list(map(int,str(num))) = [2,3,6,7]

sum(list(map(int(str(num)))) = 18

**18、僅支援關鍵字參數（kwargs）的函數**

當需要函數提供（強制）更清晰的參數時，創建僅支援關鍵字參數的函數，可能會挺有用：

def test(\*, a, b):

pass

test("value for a", "value for b") # TypeError: test() takes 0 positional arguments...

test(a="value", b="value 2") # Works...

如上所見，可以在關鍵字參數之前，放置單個 \* 參數來輕鬆解決此問題，如果我們將位置參數放在 \* 參數之前，則顯然也可以有位置參數。

**19、使用slice函數命名切片**

使用大量硬編碼的索引值會很快搞亂維護性和可讀性，一種做法是對所有索引值使用常量，但是我們可以做得更好：

# ID First Name Last Name

line\_record = "2 John Smith"

ID = slice(0, 8)

FIRST\_NAME = slice(9, 21)

LAST\_NAME = slice(22, 27)

name = f"{line\_record[FIRST\_NAME].strip()} {line\_record[LAST\_NAME].strip()}"

# name == "John Smith"

在此例中，我們可以避免神秘的索引，方法是先使用 slice 函數命名它們，然後再使用它們，還可以通過 .start、.stop和 .stop 屬性，來瞭解 slice 物件的更多資訊。

**20、在運行時提示使用者輸入密碼**

許多命令列工具或腳本需要用戶名和密碼才能操作。因此，如果你碰巧寫了這樣的程式，你可能會發現 getpass 模組很有用：

import getpass

user = getpass.getuser()

password = getpass.getpass()

# Do Stuff...

這個非常簡單的包通過提取當前用戶的登錄名，可以提示使用者輸入密碼，但是須注意，並非每個系統都支援隱藏密碼。Python 會嘗試警告你，因此切記在命令列中閱讀警告資訊。

**21、用\_\_slots\_\_節省記憶體**

如果你曾經編寫過一個程式，該程式創建了某個類的大量實例，那麼你的程式突然就會需要大量記憶體。那是因為 Python 使用字典來表示類實例的屬性，這能使其速度變快，但記憶體不是很高效。

通常這不是個問題，但是，如果你的程式遇到了問題，你可以嘗試使用\_\_slots\_\_ ：

class Person:

\_\_slots\_\_ = ["first\_name", "last\_name", "phone"]

def \_\_init\_\_(self, first\_name, last\_name, phone):

self.first\_name = first\_name

self.last\_name = last\_name

self.phone = phone

這裡發生的是，當我們定義\_\_slots\_\_屬性時，Python 使用固定大小的小型陣列，而不是字典，這大大減少了每個實例所需的記憶體。

使用\_\_slots\_\_還有一些缺點——我們無法聲明任何新的屬性，並且只能使用在\_\_slots\_\_中的屬性。  
同樣，帶有\_\_slots\_\_的類不能使用多重繼承。

**22、限制CPU和記憶體使用量**

如果不是想優化程式記憶體或 CPU 使用率，而是想直接將其限制為某個固定數字，那麼 Python 也有一個庫能做到：

import signal

import resource

import os

# To Limit CPU time

def time\_exceeded(signo, frame):

print("CPU exceeded...")

raise SystemExit(1)

def set\_max\_runtime(seconds):

# Install the signal handler and set a resource limit

soft, hard = resource.getrlimit(resource.RLIMIT\_CPU)

resource.setrlimit(resource.RLIMIT\_CPU, (seconds, hard))

signal.signal(signal.SIGXCPU, time\_exceeded)

# To limit memory usage

def set\_max\_memory(size):

soft, hard = resource.getrlimit(resource.RLIMIT\_AS)

resource.setrlimit(resource.RLIMIT\_AS, (size, hard))

我們可以看到兩個選項，可設置最大 CPU 執行時間和記憶體使用上限。

對於 CPU 限制，我們首先獲取該特定資源（RLIMIT\_CPU）的軟限制和硬限制，然後通過參數指定的秒數和先前獲取的硬限制來設置它。

最後，如果超過 CPU 時間，我們將註冊令系統退出的信號。至於記憶體，我們再次獲取軟限制和硬限制，並使用帶有 size 參數的setrlimit 和獲取的硬限制對其進行設置。

**下面重點介紹幾個和反覆運算相關的使用技巧，可以方便提升大家的工作效率。**

**很多人學Python搞不清楚方向，不同目的，你學習的側重點和難易程度都不同，必須要有針對性、選擇性地學！這樣也能提高你自己的學習效率。**

[Python學習的方法687 贊同 · 7 評論回答](https://www.zhihu.com/question/451604793/answer/1850553245)

Iterables是一個需要我們牢記的概念，因為接下來我們展示的許多技巧都使用itertools包。itertools模組提供了一些函數，用於接收Iterable物件，而不僅僅是列印逐個對象。

iterables的示例包括：

* 所有序列類型（如list、str和tuple）
* 一些非序列類型，如dict、檔物件以及類的實現中定義了\_\_iter\_\_()方法

在工作學習中，我們經常會需要使用一個簡單的函數來實現從一個list來生成新的list、set或dict，此時我們就會用到iterables概念。

**23、舉例來說：**

生成List：

names = ['John', 'Bard', 'Jessica' 'Andres']

lower\_names = [name.lower() for name in names]

生成Set：

names = ['John', 'Bard', 'Jessica' 'Andres']

lower\_names = {name.lower() for name in names}

生成Dict：

names = ['John', 'Bard', 'Jessica' 'Andres']

lower\_names = {name:name.lower() for name in names}

個人建議：

僅當for語句、函式呼叫和方法調用的數量較少時使用

**24、有時我們需要獲得兩個清單物件之間的所有可能組合，我們可能首先想到的是：**

l1 = [1, 2, 3]

l2 = [4, 5, 6]

combinations = []

for e1 in l1:

for e2 in l2:

combinations.append((e1, e2))

或者簡化一下

combinations = [(e1, e2) for e1 in l1 for e2 in l1]

上述實現已經很簡潔了，但標準庫itertools提供product函數，從而提供了相同的結果。

from itertools import product

l1 = [1, 2, 3]

l2 = [4, 5, 6]

combinatios = product(l1, l2)

**25、假設有一個元素清單，我們需要在每對相鄰元素之間比較或應用一些操作，這有時稱為2個元素的滑動視窗。**

可以採用以下方式：

from itertools import tee

from typing import Iterable

def window2(iterable: Iterable):

it, offset = tee(iter(iterable))

next(offset)

return zip(it, offset)

l = [1, 2, 3, 4, 5, 6]

dd = window2(l)

for a in dd:

print(a)

運行結果：

(1, 2)

(2, 3)

(3, 4)

(4, 5)

(5, 6)

**26、當需要一個類來存儲資訊，又覺得創建一個類並定義其\_\_init\_\_()函數太麻煩，不妨選擇使用dataclass。**

from dataclasses import dataclass

@dataclass

class Person:

name: str

age: int

address: str

上述代碼創建了一個具有預設構造函數的類，該類以與聲明相同的順序接收相應欄位的賦值。

person = Person(name='John', age=12, address='nanjing street')

dataclass的另一個優點是，預設情況下，會生成特殊方法，如\_\_str\_\_、**repr**、\_\_eq\_\_等。

注意：dataclasses構造物件時並不執行資料類型的檢查

**27、假如我們有一個dataclass，需要驗證輸入資料是否符合類型注釋。**

在這種情況下，安裝協力廠商套裝軟體pydantic並將from dataclasses import dataclass 替換為 from pydantic.dataclasses import dataclass 即可。

from pydantic.dataclasses import dataclass

@dataclass

class Person:

name: str

age: int

address: str

這將生成一個類，該類具有根據成員變數聲明的類型進行輸入資料的解析和類型驗證。Pydantic在運行時強制執行類型提示，並在資料無效時提供友好的錯誤提醒。

**28、如果我們對兩個list中的元素對做相應的函數處理**

我們最容易想到的方法：

l1 = [1, 2, 3]

l2 = [4, 5, 6]

for (e1, e2) in zip(l1, l2):

f(e1, e2)

但使用函數map可以讓代碼更加簡潔一些；

l1 = [1, 2, 3]

l2 = [4, 5, 6]

map(f, l1, l2)

**29、從一個list中隨機選擇一個元素**

此時我們使用random.choice

from random import choice

l = [1, 2, 3]

random = choice(l)

如果需要隨機選擇多個元素呢？當然是使用***random.choices***

from random import choices

l = [1, 2, 3, 4, 5]

random\_elements = choices(l, k=3)

代碼中的參數k為我們隨機選擇元素的個數；

**30、跳過可反覆運算物件的開始**

有時候你必須處理某些檔，它們以可變數量的不需要的行（例如注釋）為開頭。

string\_from\_file = """

// Author: ...

// License: ...

//

// Date: ...

Actual content...

"""

import itertools

for line in itertools.dropwhile(lambda line:line.startswith("//"), string\_from\_file.split("\n")):

print(line)

## 一、根據條件在序列中篩選資料

* 假設有一個數字清單 data, 過濾列表中的負數

data = [1, 2, 3, 4, -5]

# 使用列表推導式

result = [i for i in data if i >= 0]

# 使用 fliter 過濾函數

result = filter(lambda x: x >= 0, data)

* 學生的數學分數以字典形式存儲，篩選其中分數大於 80 分的同學

from random import randint

d = {x: randint(50, 100) for x in range(1, 21)}

r = {k: v for k, v in d.items() if v > 80}

## 二、對字典的鍵值對進行翻轉

* 使用 zip() 函數

zip() 函數用於將可反覆運算的物件作為參數，將物件中對應的元素打包成一個個元組，然後返回由這些元組組成的列表。

from random import randint, sample

s1 = {x: randint(1, 4) for x in sample("abfcdrg", randint(1, 5))}

d = {k: v for k, v in zip(s1.values(), s1.keys())}

## 三、統計序列中元素出現的頻度

* 某隨機序列中，找到出現次數最高的3個元素，它們出現的次數是多少

**方法1:**

# 可以使用字典來統計，以清單中的資料為鍵，以出現的次數為值

from random import randint

# 構造隨機序列

data = [randint(0, 20) for \_ in range(30)]

# 清單中出現數字出現的次數

d = dict.fromkeys(data, 0)

for v in d:

d[v] += 1

**方法2：**

# 直接使用 collections 模組下面的 Counter 物件

from collections import Counter

from random import randint

data = [randint(0, 20) for \_ in range(30)]

c2 = Counter(data)

# 查詢元素出現次數

c2[14]

# 統計頻度出現最高的3個數

c2.most\_common(3)

* 對某英文文章單詞進行統計，找到出現次數最高的單詞以及出現的次數

import re

from collections import Counter

# 統計某個文章中英文單詞的詞頻

with open("test.txt", "r", encoding="utf-8") as f:

d = f.read()

# 所有的單字清單

total = re.split("\W+", d)

result = Counter(total)

print(result.most\_common(10))

## 四、根據字典中值的大小，對字典中的項進行排序

* 比如班級中學生的數學成績以字典的形式存儲，請按數學成績從高到底進行排序

**方法1:**

# 利用 zip 將字典轉化為元組，再用 sorted 進行排序

from random import randint

data = {x: randint(60, 100) for x in "xyzfafs"}

sorted(data)

data = sorted(zip(data.values(), data.keys()))

**方法2:**

# 利用 sorted 函數的 key 參數

from random import randint

data = {x: randint(60, 100) for x in "xyzfafs"}

data.items()

sorted(data.items(), key=lambda x: x[1])

## 五、在多個字典中找到公共鍵

* 實際場景：在足球聯賽中，統計每輪比賽都有進球的球員

第一輪：{"C羅": 1, "蘇亞雷斯":2, "托雷斯": 1..}

第二輪：{"內瑪律": 1, "梅西":2, "姆巴佩": 3..}

第三輪：{"姆巴佩": 2, "C羅":2, "內瑪律": 1..}

from random import randint, sample

from functools import reduce

# 模擬隨機的進球球員和進球數

s1 = {x: randint(1, 4) for x in sample("abfcdrg", randint(1, 5))}

s2 = {x: randint(1, 4) for x in sample("abfcdrg", randint(1, 5))}

s3 = {x: randint(1, 4) for x in sample("abfcdrg", randint(1, 5))}

# 首先獲取字典的 keys，然後取每輪比賽 key 的交集。由於比賽輪次數是不定的，所以使用 map 來批量操作

# map(dict.keys, [s1, s2, s3])

# 然後一直累積取其交集，使用 reduce 函數

reduce(lambda x, y: x & y, map(dict.keys, [s1, s2, s3]))

## List

Python 的清單 List 基本就是其它語言的 Array.

### Initialization 初始化

List 的初始化一般用 List comprehension，往往能一行解決問題

*# 1d array*

l **=** [0 **for** \_ **in** range(len(array)]

*# or*

l **=** [0] **\*** len(array)

*# 2d*

l **=** [[0] **for** i **in** range(cols) **for** j **in** range(rows)]

### Start from the behind

你可以輕鬆從後往前訪問：

lastElement = l[-1]

lastTwo = l[-2:]

for i in range(0, -10, -1)

# 0, -1, -2, -3, -4, -5, -6, -7, -8, -9

### copy 複製

### shallow copy 淺拷貝

l2 = l1[:]

# or

l2 = l1.copy()

淺複製的問題在於，如果 l1 內部還有 list，那麼這種嵌套的索引不能被複製，比如：

a = [1, 2, [3, 4]]

b = a[:]

a[2].append(5)

print(b)

# [1, 2, [3, 4, 5]]

### deep copy 深拷貝

所以如果要做深拷貝，要節制自帶庫 copy

import copy

copy.deepcopy()

### enumerate 枚舉

當我們需要枚舉一個陣列並同時獲得值與 index 的時候可以使用：

l = ["a", "b", "c"]

for i, v in enumerate(l):

print(i, v)

# 0 a

# 1 b

# 2 c

### zip

zip 本意就是拉鍊，可以想像成將兩個陣列像拉鍊一樣挨個聚合：

>>> x = [1, 2, 3]

>>> y = [4, 5, 6]

>>> zipped = zip(x, y)

>>> list(zipped)

[(1, 4), (2, 5), (3, 6)]

### reduce

reduce 可以分別對相鄰元素使用同一種計算規則，同時每一步結果作為下一步的參數，很典型的函數式程式設計用法。

# importing functools for reduce()

import functools

# initializing list

lis = [ 1, 3, 5, 6, 2, ]

# using reduce to compute sum of list

print ("The sum of the list elements is : ",end="")

print (functools.reduce(lambda a,b : a+b,lis))

# The sum of the list elements is : 17

### map

可以將參數一一映射來計算， 比如

date = "2019-8-15"

Y, M, D = map(int, date.split('-'))

# Y = 2019, M = 8, D = 15

## deque

list 刪除末尾的操作是O(1)的，但是刪除頭操作就是O(n)，這時候我們就需要一個雙端佇列 deque。首尾的常規操作為：

* append，添加到末尾
* appendleft, 添加到開頭
* pop, 剔除末尾
* popleft，移除開頭

## sorted

list 自身有自帶的 sort(), 但是它不返回新的 list. sorted 能返回一個新的 list, 並且支持傳入參數reverse。

比如我們有一個 tuple 的陣列，我們想按照 tuple 的第一個元素進行排序：

l1 = [(1,2), (0,1), (3,10) ]

l2 = sorted(l1, key=lambda x: x[0])

# l2 = [(0, 1), (1, 2), (3, 10)]

這裡的 key 允許傳入一個自訂參數，也可以用自帶函數進行比較，比如在一個 string 陣列裡只想比較小寫，可以傳入key=str.lower

l1 = ["banana","APPLE", "Watermelon"]

l2 = sorted(l1, key=str.lower)

print(l2)

# ['APPLE', 'banana', 'Watermelon']

### lambda

你注意到我們在上面使用了 lambda 來定義一個匿名函數，十分方便。如果你熟悉其它語言類似 JS 的話，可以把它理解成一個 callback 函數，參數名一一對應就行。

### cmp\_to\_key

在 python3 中，sorted 函數取消了自帶的cmp函數，需要借助functools 庫中的 cmp\_to\_key來做比較。  
比如如果要按照陣列元素的絕對值來排序：

from functools import cmp\_to\_key

def absSort(arr):

newarr = sorted(arr, key = cmp\_to\_key(sortfunc))

return newarr

def sortfunc(a, b):

if abs(a) < abs(b):

return -1

elif abs(a) > abs(b):

return 1

else:

return a - b

## set

set 的查找操作複雜度為O(1)，有時候可以替代dict 來存儲中間過程。

* add : set 的添加是 add 不是append
* remove vs discard: 都是刪除操作，區別在於remove不存在的元素會報錯，discard不會。
* union, intersection: 快速獲得並集和交集，方便一些去重操作。

## dict

字典，相當於其它語言中的map, hashtable, hashmap之類的，讀取操作也是O(1) 複雜度

### keys(), values(), items()

這三個方法可以分別獲得key, value, {key: value}的陣列。

### setdefault

這個函數經常在初始化字典時候使用，如果某個key在字典中存在，返回它的value, 否則返回你給的 default 值。比如在建一個 trie 樹的時候

node = self.root

for char in word:

node = node.setdefault(char, {})

### OrderedDict

OrderedDict 能記錄你 key 和 value 插入的順序，底層其實是一個雙向鏈表加雜湊表的實現。我們甚至可以使用move\_to\_end這樣的函數：

>>> d = OrderedDict.fromkeys('abcde')

>>> d.move\_to\_end('b')

>>> ''.join(d.keys())

'acdeb'

# 放開頭

>>> d.move\_to\_end('b', last=False)

>>> ''.join(d.keys())

'bacde'

### defaultdict

defaultdict可以很好地來解決一些初始化的問題，比如 value 是一個 list，每次需要判斷 key 是否存在的情況。這時我們可以直接定義

d = defaultdict(list)

s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('blue', 4), ('red', 1)]

for k, v in s:

d[k].append(v)

sorted(d.items())

# [('blue', [2, 4]), ('red', [1]), ('yellow', [1, 3])]

## heapq

heapq 就是 python 的 priority queue，heapq[0]即為堆頂元素。

heapq 的實現是小頂堆，如果需要一個大頂堆，常規的一個做法是把值取負存入，取出時再反轉。  
以下是借助 heapq 來實現 heapsort 的例子：

>>> def heapsort(iterable):

... h = []

... for value in iterable:

... heappush(h, value)

... return [heappop(h) for i in range(len(h))]

...

>>> heapsort([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

## bisect

python 自帶二分查找的庫，在一些不要求實現 binary search，但是借助它能加速的場景下可以直接使用。

bisect.bisect(a, x, lo=0, hi=len(a))

這裡的參數分別為 陣列，要查找的數，範圍起始點，範圍結束點

相似函數還有

* bisect.bisect\_left
* bisect.bisect\_right  
  分別返回可以插入 x 的最左和最右 index

## Counter

Counter 接受的參數可以是一個 string, 或者一個 list, mapping

>>> c = Counter() # a new, empty counter

>>> c = Counter('gallahad') # a new counter from an iterable

>>> c = Counter({'red': 4, 'blue': 2}) # a new counter from a mapping

>>> c = Counter(cats=4, dogs=8) # a new counter from keyword args

* most\_common(n)  
  可以得到出現次數最多的 n 個數：

>>> Counter('abracadabra').most\_common(3) # doctest: +SKIP

[('a', 5), ('r', 2), ('b', 2)]

## strings

### ord, char

ord 返回單個字元的 unicode:

>>> ord('a')

97

char 則是反向操作：

>>> chr(100)

'd'

### strip

移除 string 前後的字串，預設來移除空格，但是也可以給一個字串，然後會移除含有這個字串的部分：

>>> ' spacious '.strip()

'spacious'

>>> 'www.example.com'.strip('cmowz.')

'example'

### split

按照某個字串來切分，返回一個 list, 可以傳入一個參數maxsplit來限定分離數。

>>> '1,2,3'.split(',')

['1', '2', '3']

>>> '1,2,3'.split(',', maxsplit=1)

['1', '2,3']

>>> '1,2,,3,'.split(',')

['1', '2', '', '3', '']

## int/ float

### 最大, 最小 number

有時候初始化我們需要設定 Math.max() 和 Math.min(), 在 python 中分別以 float('inf') 和 float('-inf')表示

我們也可以這麼做：

import sys

#maxint

Max = sys.maxint

### 除法

在 python3 中， / 會保留浮點，相當於 float 相除，如果需要做到像 pyhton2 中的 int 相除，需要 //：

>>> 3 / 2

1.5

>>> 3 // 2

1

### 次方

在 python 中為 \*\*:

>>> 2 \*\* 10

1024

## conditions

在 python 的三項運算式(ternary operation) 與其它語言不太一樣：

res = a if condition else b

它表示如果 condition 滿足，那麼 res = a, 不然 res = b，在類 c 的語言裡即為：

res = condition ? a : b;

## any, all

any(), all()很好理解，就是字面意思，即參數中任何一個為 true 或者全部為 true 則返回 true。經常可以秀一些騷操作:  
比如 [36. Valid Sudoku](https://link.zhihu.com/?target=https%3A//leetcode.com/problems/valid-sudoku/) 這題：

class Solution:

def isValidSudoku(self, board: List[List[str]]) -> bool:

row = [[x for x in y if x != '.'] for y in board]

col = [[x for x in y if x != '.'] for y in zip(\*board)]

pal = [[board[i+m][j+n] for m in range(3) for n in range(3) if board[i+m][j+n] != '.'] for i in (0, 3, 6) for j in (0, 3, 6)]

return all(len(set(x)) == len(x) for x in (\*row, \*col, \*pal))

## itertools

這是 python 自帶的反覆運算器庫，有很多實用的、與遍歷、反覆運算相關的函數。

### permutations 排列

permutations('ABCD', 2)

# AB AC AD BA BC BD CA CB CD DA DB DC

### combinations 組合

combinations('ABCD', 2)

# AB AC AD BC BD CD

### groupby 合併

[https://leetcode.com/problems/swap-for-longest-repeated-character-substring/discuss/355852/Python-Groupby/322898](https://link.zhihu.com/?target=https%3A//leetcode.com/problems/swap-for-longest-repeated-character-substring/discuss/355852/Python-Groupby/322898)

[k for k, g in groupby('AAAABBBCCDAABBB')] --> A B C D A B

[list(g) for k, g in groupby('AAAABBBCCD')] --> AAAA BBB CC D

## functools

這個庫裡有很多高階函數，包括前面介紹到的cmp\_to\_key 以及 reduce，但是比較逆天的有 lru\_cache，即 least recently used cache. 這個 [LRU Cache](https://link.zhihu.com/?target=https%3A//leetcode.com/problems/lru-cache/)是一個常見的面試題，通常用 hashmap 和雙向鏈表來實現，python 居然直接內置了。

用法即直接作為 decorator 裝飾在要 cache 的函數上，以變數值為 key 存儲，當反復調用時直接返回計算過的值，例子如下：

### lru\_cache

[https://leetcode.com/problems/stone-game-ii/discuss/345230/Python-DP-Solution](https://link.zhihu.com/?target=https%3A//leetcode.com/problems/stone-game-ii/discuss/345230/Python-DP-Solution)

def stoneGameII(self, A: List[int]) -> int:

N = len(A)

for i in range(N - 2, -1, -1):

A[i] += A[i + 1]

from functools import lru\_cache

@lru\_cache(None)

def dp(i, m):

if i + 2 \* m >= N: return A[i]

return A[i] - min(dp(i + x, max(m, x)) for x in range(1, 2 \* m + 1))

return dp(0, 1)