

- Ex1.3Add1:List(串列)就相當於其他語言的陣列，非常重要。請舉簡單例子來說明 List 中，元素的訪問、新增、刪除(訪問串列中的值，動態新增串列元素，動態刪除串列元素)
- Ex1.3Add2:dict(字典)是高效的資料結構。至少要會:取值,加新值,移除三種操作。請舉簡單的例子來示範這三種操作。
- Ex1.4Add1:python 的控制結構,用來界定控制結構主體的代碼區塊的開頭的字符是什麼？接下來要進入縮排區，請問縮排區習慣上按鍵盤上的那個鍵？或相當於是幾個空白？
- Ex1.4Add2:在 python 中，每個東西都算是一個物件，除了有物件導向特性，也具有傳統的「結構化程式設計」的重要特色。結構化的三種程式控制基本架構，是那三種？
- Ex1.4Add3:在 python 的重複結構，有提供兩種寫法，請問是那兩種？並說明，要如何分辨這兩種寫法適合使用的時機？
-

-
- Ex1.5 Add1: Python 的參數傳遞是用"Call by Sharing"的機制。(a)與其他語言裡的 call by reference, or call by value 比較有何不同？如果要要 call by value 或 call by reference 的效果，在 python 要如何表達？請分別考慮參數的資料型態是(b)不可變 (immutable)，與(c)非可變對象可變(mutable)的可行寫法。
-

- Ex1.5 Add2:其實真正影響傳值效果的，並不僅是看資料型態是否"immutable"？而是改變參數的操作！例如有下列兩個簡單函數 clear_a()與 clear_b()。請問呼叫函數返回後，那個會真正

```
>>> def clear_a(x):
    x=[]

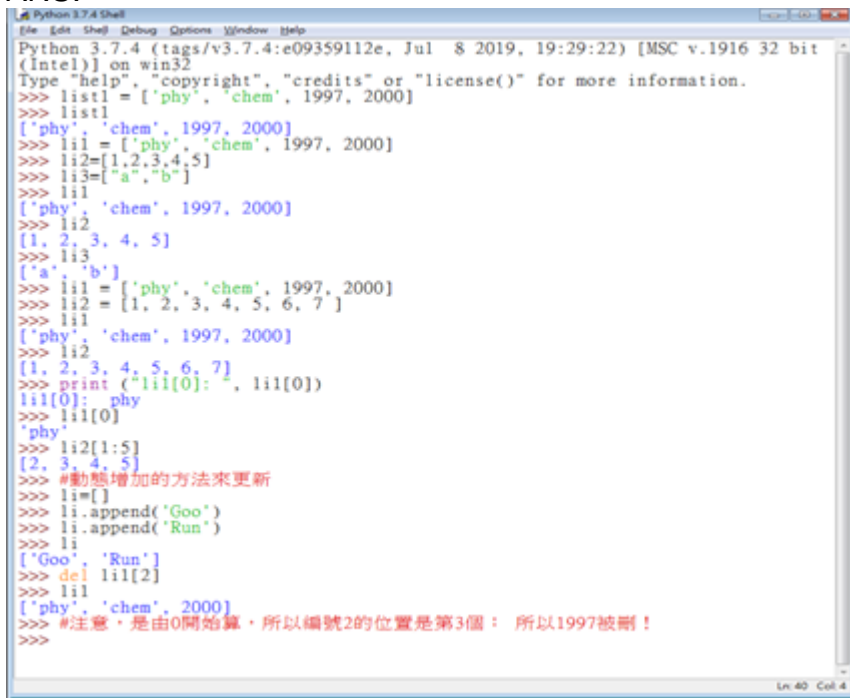
>>> def clear_b(x):
    while x: x.pop()

>>> z=[1,2,3]
>>> clear_a(z)
>>> clear_b(z)
```

把 z 清為空的 list?

- Ex1.5 Add3:python 的 built-in function 都是很重要很常用的函數。例如 print, max.. 請問，其中可以查詢字元符的 Unicode 的函數是？可以把 Unicode 轉為字元符的函數是？
 - Ex1C1.17(章未):我們是否按以下方式實現了之前提過的縮放函數 Scale,它是否正常工作？
def scale(data, factor):
 for val in data:
 val * = factor
解釋為什麼能或者為什麼不能正常工作。
 -
-

p2/7 ex_dataViz_pyjs_ch01(2024au)_ex1 ExerciseSolution (2024au) By Jiun-Ting Jiang
Ex1.3Add1:**List** 就相當於其他語言的陣列，非常重要。請舉簡單例子來說明 **List** 中，元素的訪問、新增、刪除(訪問串列中的值，動態新增串列元素，動態刪除串列元素)
ANS:

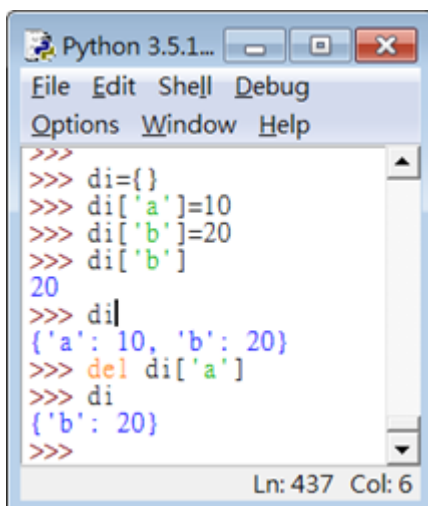


```
Python 3.7.4 Shell
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22) [MSC v.1916 32 bit
(intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> list1 = ['phy', 'chem', 1997, 2000]
>>> list1
['phy', 'chem', 1997, 2000]
>>> li1 = ['phy', 'chem', 1997, 2000]
>>> li2 = [1, 2, 3, 4, 5]
>>> li3 = ['a', 'b']
>>> li1
['phy', 'chem', 1997, 2000]
>>> li2
[1, 2, 3, 4, 5]
>>> li3
['a', 'b']
>>> li1 = ['phy', 'chem', 1997, 2000]
>>> li2 = [1, 2, 3, 4, 5, 6, 7]
>>> li1
['phy', 'chem', 1997, 2000]
>>> li2
[1, 2, 3, 4, 5, 6, 7]
>>> print ("li1[0]:", li1[0], "li2[0]:", li2[0])
li1[0]: phy
li2[0]: 1
>>> li1[0] = 'phy'
>>> li2[1:5]
[2, 3, 4, 5]
>>> #動態增加的方法來更新
>>> li = []
>>> li.append('Goo')
>>> li.append('Run')
>>> li
['Goo', 'Run']
>>> del li[2]
>>> li
['Goo', 'Run']
>>> li1 = ['phy', 'chem', 2000]
>>> #注意，是由0開始算，所以編號2的位置是第3個：所以1997被刪！
>>>
```

Ex1.3Add2:dict 字典是高效的資料結構。至少要會:取值,加新值,移除三種操作。請舉簡單的例子來示範這三種操作。

Ans:

```
>>> di={}
>>> di['a']=10
>>> di['b']=20
>>> di['b']
20
>>> di
{'a': 10, 'b': 20}
>>> del di['a']
>>> di
{'b': 20}
>>>
```



```
Python 3.5.1 Shell
File Edit Shell Debug
Options Window Help
>>>
>>> di={}
>>> di['a']=10
>>> di['b']=20
>>> di['b']
20
>>> di
{'a': 10, 'b': 20}
>>> del di['a']
>>> di
{'b': 20}
>>>
```

Ex1.4Add1:python 的控制結構,用來界定控制結構主體的代碼區塊的開頭的字符是什麼？接下來要進入縮排區，請問縮排區習慣上按鍵盤上的那個鍵？或相當於是幾個空白？

Ans:

對於所有 python 的控制結構而言，是用“:” 冒號字符，來界定控制結構主體的代碼區塊的開頭。接下來，從冒號之後的行開始進入縮排區塊。縮排區習慣上按鍵盤上“Tab”鍵即可，相當於是 4 個空白。

Ex1.4Add2:在 python 中，每個東西都算是一個物件，除了有物件導向特性，也具有傳統的「結構化程式設計」的重要特色。結構化的三種程式控制基本架構，是那三種？

Ans:

循序結構、選擇結構(條件結構)、重覆結構。

Ex1.4Add3:在 python 提供兩種重複結構是那兩種？請說明，要如何分辨適合使用的時機？

Ans:

(a)兩種重複結構是 For 與 While,

(b)分辨適合的使用時機：

(b1)For Loop, 適用於計數式的條件控制。使用者已知道迴圈的次數，或者知道重複的最大次數。

(b2)While Loop, 迴圈次數未知，必須滿足特定條件運算式，才能進入迴圈，反之，條件不成立，就結束迴圈。

Ex1.5 Add1: Python 的參數傳遞是用“Call by Sharing”的機制。(a)與其他語言裡的 call by reference, or call by value 比較有何不同？(b)如果要要 call by value 或 call by reference 的效果，在 python 要如何表達？請分別考慮參數的資料型態是(b1)可變(mutable)，與(b2)不可變 (immutable)的可行寫法。(c)另外，考慮函數呼叫的特性，若想 call by reference 的「改變」效果，又「不想用可變(mutable)資料型態」時，是否有可行的寫法？

Ans:

python 參數傳遞是用“Call by Sharing“

(a)

(a1)為何不是 call by value，因為傳入的參數，並「不複製對象(not copy)」，這樣的好處是即使在參數或返回值是複雜對象的情況下，也可以確保有效地調用函數。

(a2)為何不是 call by reference，因為傳入不可變(immutable)的參數時。則函數內的改變並不會影響原參數！

所以，python 的參數傳遞稱為 Call by sharing (或稱為“Call-by-Object”,也可稱為"Call by Object Reference“)

(b) 在 python 要如何簡單表達這兩種參數傳遞法：

(b1)如果想要有 call by value 的效果，就把參數選擇不可變的(immutable)資料型態(例如 int, str, tuple) 即可。

(b2)如果想要有 call by reference 的效果，就把參數選擇可變的(mutable)資料型態(例如 list) 即可。

(c)另外，考慮函數呼叫的特性。可把要被改變的變數放在呼叫函數時等號的左邊，就可藉著(tuple)的傳回方式來改變等號左邊為新值！

參考：

```

1 def scale(data, factor): #data有call by ref的效果
2     for j in range(len(data)):
3         data[j] *= factor
4 a=[1,3,5,10]
5 scale(a,100)
6 a
7 #[100, 300, 500, 1000]

```

```
: [100, 300, 500, 1000]
```

```

>>> def scale(data, factor): #data 有 call by ref 的效果
      for j in range(len(data)):
          data[j] *= factor

```

```

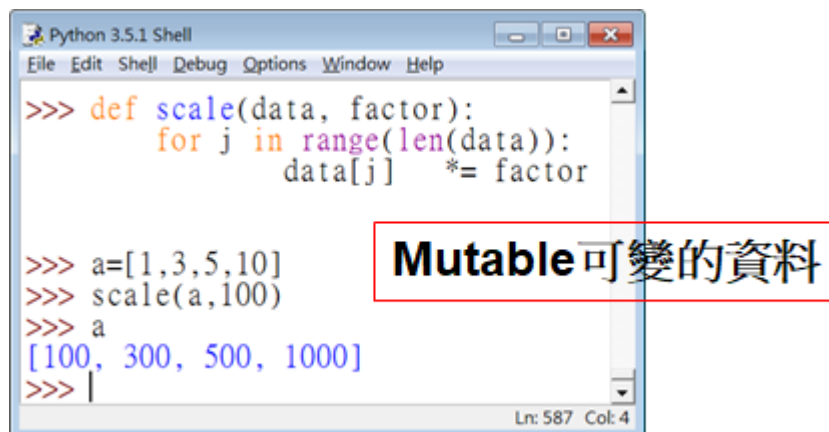
>>> a=[1,3,5,10]
>>> scale(a,100)
>>> a
[100, 300, 500, 1000]
>>>

```

Ex1.5 Add2:其實真正影響傳值效果的，並不僅是看資料型態是否“immutable”?而是改變參數的操作！例如有下列兩個簡單函數 `clear_a()`與 `clear_b()`。請問呼叫後的結果，那個會真正把 `z` 清為空的 list?

Ans:

參考圖。



```

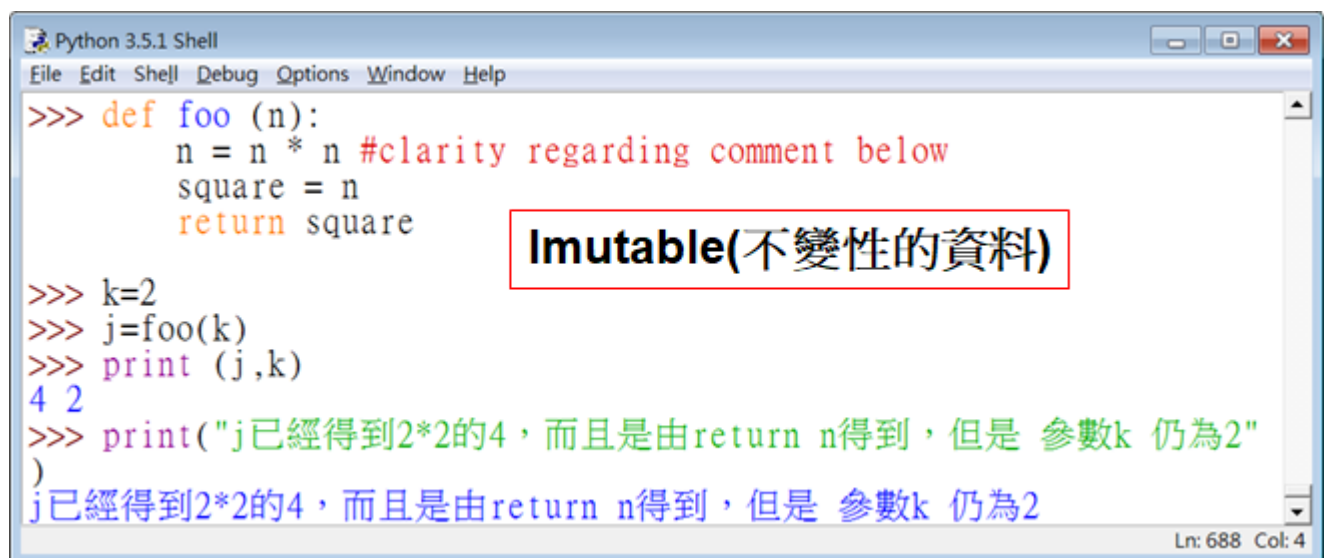
Python 3.5.1 Shell
File Edit Shell Debug Options Window Help

>>> def scale(data, factor):
      for j in range(len(data)):
          data[j] *= factor

>>> a=[1,3,5,10]
>>> scale(a,100)
>>> a
[100, 300, 500, 1000]
>>> |

```

Mutable可變的資料



```

Python 3.5.1 Shell
File Edit Shell Debug Options Window Help

>>> def foo (n):
      n = n * n #clarity regarding comment below
      square = n
      return square

>>> k=2
>>> j=foo(k)
>>> print (j,k)
4 2
>>> print("j已經得到2*2的4，而且是由return n得到，但是 參數k 仍為2"
)
j已經得到2*2的4，而且是由return n得到，但是 參數k 仍為2

```

Immutable(不變性的資料)

```

Python 3.5.1 Shell
File Edit Shell Debug Options Window Help

>>> def clear_a(x):
>>>     x=[]

>>> def clear_b(x):
>>>     while x: x.pop()

>>> z=[1,2,3]
>>> clear_a(z)
>>> z
[1, 2, 3]
>>> clear_b(z)
>>> z
[]
>>>
Ln: 799 Col: 10

```

`clear_b()`，則會真的會把原來的 `list` 的內容，全 `pop` 出來，而清為空白。

但是 `clear_a()`，在返回後，並沒改變 `a` 原來的內容！

因為在函數內的 `x=[]`，只會改變別名的新配到的內容，而不會更改實際參數！

```

: 1 def foo(n):
2     n=n*n
3     square=n
4     return square
5 k=2
6 j=foo(k)
7 print(j,k)
8 print('j有正確得到平方，而且n也維持原來的2')

```

4 2

j有正確得到平方，而且n也維持原來的2

```

: 1 def clear_a(x):
2     x=[]
3 def clear_b(x):
4     while x: x.pop()
5 z=[1,2,3]
6 clear_a(z)
7 z2=[1,2,3]
8 clear_b(z2)
9 z,z2

```

: ([1, 2, 3], [])

p6/7 ex_dataViz_pyjs_ch01(2024au)_ex1 ExerciseSolution (2024au) By Jiun-Ting Jiang
Ex1.5 Add3:python 的 built-in function 都是很常用的函數。例如 print, max.. 其中可以查詢字元符的 Unicode 的函數是？可以把 Unicode 轉為字元符的函數是？

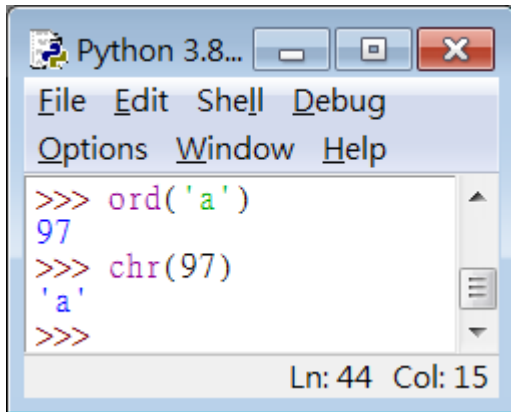
Ans:

這兩個是互為反函數：

a)查詢字元符的 Unicode 的函數是 chr(integer)

b)可以把 Unicode 轉為字元符的函數是 ord(char)

請參考下面實驗照相確認：



Ex1C1.17(章末):我們是否按以下方式實現了之前提過的縮放函數 Scale,它是否正常工作？

```
def scale(data, factor):
```

```
    for val in data:
```

```
        val * = factor
```

解釋為什麼能或者為什麼不能正常工作？

Ans:

以下列程式來測試：

#-----Ex1-C1-17(章末)-----

```
def scale(data, factor): #錯誤設計
```

```
    for val in data:
```

```
        val *= factor
```

```
print('bad--錯誤設計，而沒有放大！')
```

```
data = [1,2,3,4,5]; print (data)
```

```
scale(data, 5); print (data)
```

```
def realscale(data, factor): #正確設計，有放大！
```

```
    for i in range (len(data)):
```

```
        data[i]*=factor    #注意 data*=factor #這會將陣列與其自身連接多次！
```

```
print ('\nGood--可正確放大 list 的元素！')
```

```
data = [1,2,3,4,5]; print (data)
```

```
realscale(data, 5); print (data)
```

解釋：

(bad_錯誤設計) val * = factor 會創建 val 的新實例，所以不會改原始對象物件的 data[]

(good_正確設計) 因為 data [i]會引用 data 編號 i 元素，這將可以更改原始數組的內容。

在列表 data[i] 中更改對元素 i 的引用，這會更改原始數組

```
|: 1 #-----Ex1-C1-17(章末)-----
2 def scale(data, factor):#錯誤設計
3     for val in data:
4         val *= factor
5     print('bad--錯誤設計，而沒有放大！')
6     data = [1,2,3,4,5]; print (data)
7     scale(data, 5); print (data)
8
9
10 def realscale(data, factor):#正確設計，有放大！
11     for i in range (len(data)):
12         data[i]*=factor
13     print ('\nGood--可正確放大list的元素！')
14     data = [1,2,3,4,5]; print (data)
15     realscale(data, 5); print (data)
16     """
17     (1)val *= Factor 建立 val 的新實例，但不會變更原始物件的引用
18     (2)在列表 data[i] 中更改對元素 i 的引用，這會更改原始數組
19
20     """
```

bad--錯誤設計，而沒有放大！

[1, 2, 3, 4, 5]

[1, 2, 3, 4, 5]

Good--可正確放大list的元素！

[1, 2, 3, 4, 5]

[5, 10, 15, 20, 25]
