3.1. List

Fig. 3.1.1 Photo by Glenn Carstens-Peters on Unsplash

Note

Outline

Note

Roadmap

1. This topic: List

> myMaze, myBoard, myBlock
>
> | Container | | | | |
> |---|---|---|---|---|
> | tuple Statement | list Statement | set Statement | dict Statement | del Statement* |

2. Course: Python 1
3. Subject: Programming
4. Field
    a. Software Engineering (SE)
    b. Computer Science and Information Engineering (CSIE)
    c. Electrical/Electronics Engineering (EE)

3.1.1. Introduction

1. 串列（list）會在[] 裡面放置資料，而這些資料是有順序的，且具有可以更改的型態。
2. 值得注意的是，tuple 跟list 很相似，差異點就在於list 可以彈性的增加、刪除與修改，tuple 則不行。
3. tuple 就像是客製化的模板，你可以隨意設定這個tuple 要多大，裡面要裝甚麼東西；但是一旦建立好就不能再更動，反之，list 就會比tuple 更加具有更動的彈性。
4. 既然list 具有的彈性較高，又可以修改資料內容，那為何還要有tuple 的存在呢？
5. 因為tuple 雖然沒有append、extend、remove 這些方法，但是因為不能修改，所以在存取的速度上比起list 要來得快。
6. 此外，由於tuple 屬於不可變動值，所以可以拿來當作字典（dict）1 的key，故在使用上還是有list 所無法取代的部分。在這裡列出一些常用的list 的函數：

3.1.1.1. Ex1

1. Source

> **Listing 3.1.1.1.1 /src/List/p0702List.py**

```
'''
Created on 2015年9月1日

@author: cph
'''
a = []                          # 建立空串列
b = [1, 2.0, "3", [4], (5), (6, )]     # 建立具有五個不同型態的元素
print(a)
print(b[1])                     # 印出b 的第1 個元素 2.0
print(b[3])                     # 印出b 的第3 個元素 [4]
print(b)                        # 印出b
print(b[-1])
print(b[9])
```

2. Output

```
[]
2.0
[4]
[1, 2.0, '3', [4], 5, (6, )]
(6,)
Traceback (most recent call last):
  File
"W:\_D\Data.cph\Workspace\Python3\Py8.Doc.WhoNotes.Code.Sphinx.Gitla
line 12, in <module>
    print(b[9])
IndexError: list index out of range
```

3. Note that the (5) inside the list a is an integer, that is, the Python treats it as an integer, not a tuple. Hence, if we want to delcare a tuple data, we have to use (6,) type.

3.1.1.2. Ex2a: Rail Fence: 2

1. Method

```
Hello, Python...

H l o   y h n .
 e l , P t o . .
```

See also

Rail Fence Cipher

a. Rail fence cipher, Wikipedia, 20230726. [Web] [Web: 中文]

2. Code+Output: N = 2

Ex2a.py Output

Listing 3.1.1.2.2 /src/List/Ex2a-CipherRailFence/output.txt

```
Encry: Hlo yhn.el,Pto..
Decry: Hello, Python...
Encry:
Decry:
Encry: 1
Decry: 1
Encry: 1a
Decry: 1a
Encry: 12a
Decry: 1a2
Encry: Hlo 喜Pto程設..的常el,我歡yhn式計.真非棒
Decry: Hello, 我喜歡Python程式設計...真的非常棒
```

3.1.1.3. Ex2b: Rail Fence: N

1. N = Any Positive Integer

```
Hello, Python...

H           n
 e     P     .
  l     y     .
   l     t     .
    o     h
     ,     o
```

2. Code+Output

Ex2b.py Output

Listing 3.1.1.3.2 /src/List/Ex2b-CipherRailFence/6.txt

```
Input positive integer (n>=1): 6
Encry: H neP.ly.lt.oh ,o
Decry: Hello, Python...
Encry:
Decry:
Encry: 1
Decry: 1
Encry: 1a
Decry: 1a
Encry: 1a2
Decry: 1a2
```

```
Encry: H t設的e我h計非l喜o.常l歡n.棒oP程. ,y式真
Decry: Hello, 我喜歡Python程式設計...真的非常棒
```

3.1.1.4. Ex2c: Rail Fence: W

1. N = Any Positive Integer
2. Another cipher type, we named as W style.

```
Level = 3
Data  = Hello, Python~~~
Encry = [Hoyn] [el,Pto~~] [l h~]

H...o...y...n...
.e.l.,.P.t.o.~.~
..l... ...h...~.
```

```
Level = 4
Data  = 1234567890123456789
Encry = [1739] [268248  ] [359157  ] [406  ]

1.....7.....3.....9.....
.2...6.8...2.4...8. ...
..3.5...9.1...5.7... . .
...4.....0.....6..... ..
```

3. Code+Output

Ex2c.py | Output: n = 2 | Output: n = 3 | Output: n = 4 | Output: n = 5

Listing 3.1.1.4.5 /src/List/Ex2c-CipherRailFence/5.txt

```
Positive integer (n>=2): 5
Origin: [1234567890123456789]
Encry : [19728068 37159 4624  53 ]
Decry : [1234567890123456789]

Origin: [123456789012345678]
Encry : [19728068 3715  4624  53 ]
Decry : [123456789012345678]

Origin: [12345678901234567]
Encry : [1972806  3715  4624  53 ]
Decry : [12345678901234567]

Origin: [Hello, Python...]
Encry : [HyePt.l h.l,o.on]
Decry : [Hello, Python...]

Origin: []
```

```
Encry : []
Decry : []

Origin: [1]
Encry : [1       ]
Decry : [1]

Origin: [1a]
Encry : [1a      ]
Decry : [1a]

Origin: [1a2b]
Encry : [1a 2 b  ]
Decry : [1a2b]

Origin: [Hello, 我喜歡Python程式設計~~~真的非常棒]
Encry : [H喜程的e我歡n式真非 l Po設~常 l,yh計~棒 ot~ ]
Decry : [Hello, 我喜歡Python程式設計~~~真的非常棒]
```

3.1.1.5. Ex2d: Rail Fence: W Final

1. N = Any Positive Integer
2. Another cipher type, we named as W style.
3. We refactor the codes.
4. Code+Output

Ex2d.py | Output: n = 4 | Output: n = 7

Listing 3.1.1.5.3 /src/List/Ex2d-CipherRailFence/7.txt

```
Positive integer (n>=2): 7
Origin: [1234567890123456789]
Encry.: [13224 315 406 597 688 79]
Decry.: [1234567890123456789]
Match?: Success

Origin: [123456789012345678]
Encry.: [13224 315 406 597 688 7 ]
Decry.: [123456789012345678]
Match?: Success

Origin: [12345678901234567]
Encry.: [13224 315 406 597 68  7 ]
Decry.: [12345678901234567]
Match?: Success

Origin: [Hello, Python...]
Encry.: [Hneo. lh. lt. oy  ,P    ]
Decry.: [Hello, Python...]
```

```
Match?: Success

Origin: []
Encry.: []
Decry.: []
Match?: Success

Origin: [1]
Encry.: [1           ]
Decry.: [1]
Match?: Success

Origin: [1a]
Encry.: [1a          ]
Decry.: [1a]
Match?: Success

Origin: [1a2b]
Encry.: [1a 2 b      ]
Decry.: [1a2b]
Match?: Success

Origin: [Hello, 我喜歡Python程式設計~~~真的非常棒]
Encry.: [Ht的eyh真非 lPo~常 l歡n~棒 o喜程~  ,我式計    設 ]
Decry.: [Hello, 我喜歡Python程式設計~~~真的非常棒]
Match?: Success
```

3.1.2. Methods

函數	描述
list.append(object)	在list的最後面加入object。
list.extend(l)	將另一個具有順序的物件l附加在此list的最後面。
list.insert(index, object)	將元素object插入在index值為i的位置。
list.remove(value)	將list裡面的第一個符合value之元素刪除。
list.pop(i)	將list裡面index值為i的元素刪除並回傳。如果沒有給定i值，則刪除並回傳最後一個。
list.clear()	將整個list清空。
list.index(value[, start[, stop]])	查詢list當中第一個value出現的index值並回傳。
list.count(value)	計算list中value出現的次數。
list.sort()	將list裡面的元素做排序並儲存。
list.reverse()	將list裡面的元素順序顛倒並儲存。
list.copy()	將list直接複製一份。

1. 下面菲絲恩逐項介紹list 的常用函數：

3.1.2.1. Ex1: appand()

1. list.append(x)

    a. append(x) 會將圓括號裡面的物件x 加在list 的最後端。

    b. 記得，Python 裡面所有東西都是物件，因此所有東西都可以用append() 加入list 的最後端。

2. Code+Output

    a. Source

Listing 3.1.2.1.1 /src/List/p0703ListAppend.py

```
'''
Created on 20210720
@author: cph
'''
list1 = [1, 2, 3, 4]          # 給定list1 初始值
print(list1)

list2 = ["a", "b", "c"]       # 給定list2 初始值
list1.append(list2)           # 將list2 的元素用extend 逐項加到list1 最後端
print(list1)

list1.append("test")          # 將'test' 的各個字元拆，開逐項加到list1 最後端
print(list1)

list1.append(('test', ))# 將"test"的字串加到list1 最後端
print(list1)
```

    b. Output

```
[1, 2, 3, 4]
[1, 2, 3, 4, ['a', 'b', 'c']]
[1, 2, 3, 4, ['a', 'b', 'c'], 'test']
[1, 2, 3, 4, ['a', 'b', 'c'], 'test', ('test',)]
```

3.1.2.2. Ex2: extend()

1. list.extend(l)

a. 將有順序物件中的元素個別加到list 的最後端。

b. 其中，extend() 只接受有順序的物件為參數，也就是有index 值的物件，例如list、tuple、str 都可以。

c. 不過，如果以str 為參數傳入的話，會將str 的每個字元拆開當作個別元素加到list 的最後端。

2. Code+Output

a. Source

Listing 3.1.2.2.1 /src/List/p0703ListExtend.py

```
'''
Created on 2015年9月1日
@author: cph
'''
list1 = [1, 2, 3, 4]          # 給定list1 初始值
print(list1)

list2 = ["a", "b", "c"]       # 給定list2 初始值
list1.extend(list2)           # 將list2 的元素用extend 逐項加到list1 最後端
print(list1)

list1.extend("test")          # 將'test' 的各個字元拆，開逐項加到list1 最後端
print(list1)

list1.extend(('test', ))# 將"test"的字串加到list1 最後端
print(list1)
```

b. Output

```
[1, 2, 3, 4]
[1, 2, 3, 4, 'a', 'b', 'c']
[1, 2, 3, 4, 'a', 'b', 'c', 't', 'e', 's', 't']
[1, 2, 3, 4, 'a', 'b', 'c', 't', 'e', 's', 't', 'test']
```

3. Note that the “test” at line 12 as a container, because string is a kind of the containers. Hence, if we want to extend a set of strings, we have to use (‘test’,) type.

3.1.2.3. Ex3: insert()

1. list.insert(index, object)

a. insert() 需要兩個傳入參數，分別是index 跟object。

b. index 接收int 型態的值，用來指定要插入object 的位置；object 接收object 型態的資料，只要是物件都可以利用insert 插入list。

2. Code+Output

a. Source

Listing 3.1.2.3.1 /src/List/p0704ListInsert.py

```
'''
Created on 2015年9月1日
@author: cph
'''
list1 = [1, 2, 3, 4]         # 給定list1 初始值
print(list1)

list1.insert(2, "insert")    # 將'insert' 插入到index = 2 的地方
print(list1)

list1.insert(7, "insert2")   # 將'insert2' 插入到index = 7 的地方
print(list1)

list1.insert(6, "insert3")   # 將'insert3' 插入到index = 6 的地方
print(list1)

list1.insert(6, ['insert4', ])  # 將['insert4']插入到index = 6 的地方
print(list1)
```

b. Output

```
[1, 2, 3, 4]
[1, 2, 'insert', 3, 4]
[1, 2, 'insert', 3, 4, 'insert2']
[1, 2, 'insert', 3, 4, 'insert2', 'insert3']
[1, 2, 'insert', 3, 4, 'insert2', ['insert4'], 'insert3']
```

3. 在此例中，菲絲恩故意在insert() 中以不存在的index 值當作參數傳入，可以看到一個很特殊的現象。

4. 也就是，當使用 insert() 將 “insert2” 插入到index 值為7 的位置時，因為此時list1 的長度只有5，因此index = 5、6、7 的位置都是空的，而印出list1 時看不出”insert2” 是插在index = 7 的位置，只看得出插在最後端；

5. 之後再將 “insert3” 插在 index = 6 的位置，如果剛才確實是將”insert2” 插在index 值為7 的位置，那輸出結果就應該是[1, 2, ‘insert’, 3, 4, ‘insert3’, ‘insert2’] 才對。
6. 由此可知，當insert() 被傳入超出現有範圍的index 值時，只會將物件插在list 的最後端，而不會遵照使用者所指定的index 值插入。

3.1.2.4. Ex4: remove()

1. list.remove(value)
    a. value 接受所有可能的值當作傳入參數，並將list 裡面的第一個符合value之資料刪除。
    b. 由於list 屬於容器資料型別，因此可以存放所有型態的物件。
    c. 只要輸入的value 符合Python 所定義的物件表現值，就可能被list.remove()所接受。

2. Code+Output
    a. Source

Listing 3.1.2.4.1 /src/List/p0705ListRemove.py

```
'''
Created on 2015年9月1日
@author: cph
'''
list1 = [1, "a", (12.4), (12.5, ), [True], range(10), 1]
# 將不同型別的物件放入list1
print(list1)

list1.remove(1)             # 將1 從list1 移除
print(list1)

list1.remove("a")           # 將‘a’ 從list1 移除
print(list1)

list1.remove(range(10))     # 將range(10) 從list1 移除
print(list1)

list1.remove(12.4)          # 將(12.4) 從list1 移除
print(list1)

list1.remove([True])        # 將[True] 從list1 移除
print(list1)

list1.remove((12.5,))       # 將(12.5, ) 從list1 移除
print(list1)
```

b. Output

```
[1, 'a', 12.4, (12.5,), [True], range(0, 10), 1]
['a', 12.4, (12.5,), [True], range(0, 10), 1]
[12.4, (12.5,), [True], range(0, 10), 1]
[12.4, (12.5,), [True], 1]
[(12.5,), [True], 1]
[(12.5,), 1]
[1]
```

3. Note that as we mentioned, the (12.4) inside the list1 is a float, NOT a tuple, that is, the Python treats it as a float, not a tuple.

3.1.2.5. Ex5: pop()

1. list.pop(i)

   a. i 接受int型態的值為參數，然後會將index值為i的元素回傳到呼叫此函數的程式碼，並且將此元素從list中移除。

2. Code+Output

   a. Source

Listing 3.1.2.5.1 /src/List/p0706ListPop.py

```
'''
Created on 20210720
@author: cph
'''
# 給定list1 初始值，為讓index 的查詢功能更明顯，給予較多值
list1 = [1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32,
1, 8, 4, 6]
print(list1)
print(list1.pop())
print(list1)
print(list1.pop())
print(list1)
print(list1.pop())
print(list1)
```

   b. Output

```
[1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32, 1, 8, 4,
6]
6
[1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32, 1, 8, 4]
```

```
4
[1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32, 1, 8]
8
[1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32, 1]
```

Important

pop(): First vs. Last

a. Code

```
import timeit

sPopFirst = '''
def popFirst():
  lData = [1, 9, 2, 8, 3, 7, 4, 6, 5, 4, 6, 3, 7, 2]

  for i in range(len(lData)):
    lData.pop(0)
'''

sPopLast = '''
def popFirst():
  lData = [1, 9, 2, 8, 3, 7, 4, 6, 5, 4, 6, 3, 7, 2]

  for i in range(len(lData)):
    lData.pop()
'''
if (__name__ == '__main__'):
  print(f'First: {timeit.timeit(sPopFirst,
number=1000000)}')
  print(f'Last : {timeit.timeit(sPopLast,
number=1000000)}')
```

b. Output

```
First: 0.14794179999989865
Last : 0.19487919699986378
```

3.1.2.6. Ex6: clear()

1. list.clear()

   a. 此函數沒有傳入參數。

   b. 呼叫此函數會將list 的元素全部清空。

2. Code+Output

a. Source

Listing 3.1.2.6.1 /src/List/p0706ListClear.py

```
'''
Created on 20210720
@author: cph
'''
# 給定list1 初始值，為讓index 的查詢功能更明顯，給予較多值
list1 = [1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32,
1, 8, 4, 6]
print(list1)
print(list1.clear())
print(list1)
```

b. Output

```
[1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32, 1, 8, 4,
6]
None
[]
```

Important

clear() vs. []

a. Code

```
import timeit

str1 = 'list1 = [1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5,
7, 4, 32, 1, 8, 4, 6]'
str2 = 'list1.clear()'
str3 = 'list1 = []'
print(timeit.timeit(str1 + ';' + str2))
print(timeit.timeit(str1 + ';' + str3))
```

b. Output

```
0.3230463929994585
0.26515769900106534
```

c. Conclustion: Compared the execution speed, [] is better than clear().

3.1.2.7. Ex7: index()

1. list.index(value[, start[, stop]])

    a. index()有三個參數，其中value是必須傳入的，規則跟remove()一樣， 只要是Python所定義的值都可以接受。

    b. start跟stop都是選擇性選項屬性，接受型態為int的值。

    c. index()會將list中符合value的第一個元素index值回傳給呼叫此函數的程式碼。

    d. 值得注意的是，不管start的給定值，所回傳的index 值都是以原本的索引順序為準。

2. Code+Output

    a. Source

Listing 3.1.2.7.1 /src/List/p0706ListIndex.py

```
'''
Created on 2015年9月1日
@author: cph
'''
# 給定list1 初始值，為讓index 的查詢功能更明顯，給予較多值
list1 = [1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32,
1, 8, 4, 6]
print(list1)
print(list1.index(2))           # 在list1 裡面找尋2，並回傳第一個2 的index 值
print(list1.index(2, 7))        # 從list1[7] 之後找尋2，並回傳第一個2 的index 值
print(list1.index(4, 5, 15))    # 在list1[5:15] 裡面找尋4，並回傳第一個4 的index 值
```

    b. Output

```
[1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32, 1, 8, 4,
6]
1
9
10
```

3.1.2.8. Ex8: count()

1. list.count(value)

a. 跟index() 與remove() 一樣，count() 接受所有Python 內部定義的值。

b. count() 會回傳在list 中和value 值相同的元素個數，並以int 型態回傳值。

2. Code+Output

a. Source

Listing 3.1.2.8.1 /src/List/p0706ListCount.py

```
'''
Created on 20210720
@author: cph
'''
# 給定list1 初始值，為讓index 的查詢功能更明顯，給予較多值
list1 = [1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32,
1, 8, 4, 6]
print(list1)
print(list1.count(1))
print(list1.count(5))
print(list1.count(14))
```

b. Output

```
[1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32, 1, 8, 4,
6]
3
2
1
```

3.1.2.9. Ex9a: sort()

1. list.sort()

a. sort() 不需要傳入參數。呼叫sort() 的list 會將其所有元素作排序。

b. 要注意的是，如果list 裡面的元素是不同型態的話，則將無法彼此進行比較。

c. 而如果是字串型別，Python 會比較字串中第一個字母的 Unicode 碼大小；如果是 list 或tuple，則會比較index = 0 的元素(作為比較的元素還是需要是同資料型態)。

2. Code+Output

a. Source

Listing 3.1.2.9.1 /src/List/p0706ListSort.py

```
'''
Created on 20210720
@author: cph
'''
# 給定list1 初始值，為讓index 的查詢功能更明顯，給予較多值
list1 = [1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32,
1, 8, 4, 6]
print(list1)
print(list1.sort())
print(list1)

list1 = [1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32,
1, 8, 4, 6]
print(list1)
list2 = sorted(list1)
print(list1)
print(list2)
```

b. Output

```
[1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32, 1, 8, 4,
6]
None
[1, 1, 1, 2, 2, 3, 4, 4, 4, 5, 5, 6, 7, 8, 14, 32, 45, 47,
65]
[1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32, 1, 8, 4,
6]
[1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32, 1, 8, 4,
6]
[1, 1, 1, 2, 2, 3, 4, 4, 4, 5, 5, 6, 7, 8, 14, 32, 45, 47,
65]
```

Important

sort() vs. sorted()

a. Code

```
import timeit

str1 = 'list1 = [1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5,
7, 4, 32, 1, 8, 4, 6]'
str2 = 'list1.sort()'
str3 = 'list2 = sorted(list1)'

print(timeit.timeit(str1 + ';' + str2))
print(timeit.timeit(str1 + ';' + str3))
```

b. Output

```
1.3136119690007035
1.5962955560007686
```

c. Conclustion: Compared the execution speed, sort() is better than sorted().

3.1.2.10. Ex9b: sort()

1. Code+Output

a. Source

Listing 3.1.2.10.1 /src/List/p0706ListSort2.py

```
'''
Created on 20210720
@author: cph
'''
# 給定list1 初始值，為讓index 的查詢功能更明顯，給予較多值
list1 = ['xo', 'AI', 'vs', 'ab', 'ik',
         'joy', 'copy', 'OK', 'pop', 'work',
         'quick', 'echo', 'voice', 'vector', 'cat',
         'mark', 'okay']
print(list1)
print(list1.sort())
print(list1)

list1 = ['xo', 'AI', 'vs', 'ab', 'ik',
         'joy', 'copy', 'OK', 'pop', 'work',
         'quick', 'echo', 'voice', 'vector', 'cat',
         'mark', 'okay']
print(list1)
```

```
list2 = sorted(list1)
print(list1)
print(list2)
```

b. Output

```
['xo', 'AI', 'vs', 'ab', 'ik', 'joy', 'copy', 'OK', 'pop',
'work', 'quick', 'echo', 'voice', 'vector', 'cat', 'mark',
'okay']
None
['AI', 'OK', 'ab', 'cat', 'copy', 'echo', 'ik', 'joy',
'mark', 'okay', 'pop', 'quick', 'vector', 'voice', 'vs',
'work', 'xo']
['xo', 'AI', 'vs', 'ab', 'ik', 'joy', 'copy', 'OK', 'pop',
'work', 'quick', 'echo', 'voice', 'vector', 'cat', 'mark',
'okay']
['xo', 'AI', 'vs', 'ab', 'ik', 'joy', 'copy', 'OK', 'pop',
'work', 'quick', 'echo', 'voice', 'vector', 'cat', 'mark',
'okay']
['AI', 'OK', 'ab', 'cat', 'copy', 'echo', 'ik', 'joy',
'mark', 'okay', 'pop', 'quick', 'vector', 'voice', 'vs',
'work', 'xo']
```

c. Note that capital characters are smaller the normal characters.

d. Please refer to the ASCII code table at Wikipedia.

Important

sort() vs. sorted()

a. Code

```
import timeit

str1 = "list1 = ['xo', 'AI', 'vs', 'ab', 'ik', 'joy',
'copy', 'OK', 'pop', 'work', 'quick', 'echo', 'voice',
'vector', 'cat', 'mark', 'okay']"
str2 = 'list1.sort()'
str3 = 'list2 = sorted(list1)'

print(timeit.timeit(str1 + ';' + str2))
print(timeit.timeit(str1 + ';' + str3))
```

b. Output

```
1.4276161210000282
1.8811156219999248
```

c. Conclustion: Compared the execution speed, sort() is better than sorted().

3.1.2.11. Ex10a: reverse()

1. list.reverse()

    a. reverse() 不需要傳入參數。

    b. 呼叫reverse() 的list 會將所有元素的順序進行反轉。

2. Code+Output

    a. Source

Listing 3.1.2.11.1 /src/List/p0706ListReverse.py

```
'''
Created on 20210720
@author: cph
'''
# 給定list1 初始值，為讓index 的查詢功能更明顯，給予較多值
list1 = [1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32,
1, 8, 4, 6]
print(list1)
print(list1.reverse())
print(list1)

list2 = sorted(list1)
print(list2)
print(list2.reverse())
print(list2)
```

    b. Output

```
[1, 2, 3, 5, 1, 47, 65, 45, 14, 2, 4, 5, 7, 4, 32, 1, 8, 4,
6]
None
[6, 4, 8, 1, 32, 4, 7, 5, 4, 2, 14, 45, 65, 47, 1, 5, 3, 2,
1]
[1, 1, 1, 2, 2, 3, 4, 4, 4, 5, 5, 6, 7, 8, 14, 32, 45, 47,
65]
None
[65, 47, 45, 32, 14, 8, 7, 6, 5, 5, 4, 4, 4, 3, 2, 2, 1, 1,
1]
```

3.1.2.12. Ex10b: reverse()

1. Code+Output

Code Output

Listing 3.1.2.12.1 /src/List/p0706ListReverse2.py

```
'''
author: CPH
since: 20230729
'''

if __name__ == '__main__':
    lsS = ['a', 'b', 'c', 'd']
    print(f'lsS Origin: {lsS}')
    print(f'lsS Type  : {type(lsS)}')

    lsS.reverse()
    print('Method 1...')
    print(f'\tlsS     : {lsS}')
    print(f'\tlsS Type: {type(lsS)}')

    print('Method 2...')
    print('\tlsS For :', end=' ')
    for s in lsS:
        print(s, end='')

    print('\nMethod 3...')
    print('\t*lsS    :', *lsS)
    print('\t*lsS Sep:', *lsS, sep='')

```

3.1.2.13. Ex11: copy()

1. list.copy()

    a. copy() 不需要傳入參數。

    b. 呼叫copy() 的程式碼會得到一個跟list 完全一樣的複製清單，但是值得注意的是，使用此函數所得的清單跟原本的list 所指向的物件並不相同。

2. Code+Output

    a. Source

Listing 3.1.2.13.1 /src/List/p0706ListCopy.py

```
'''
Created on 20210720
@author: cph
```

```
'''
# 給定stu 初始值並印出，確認stu 初始資料
stu = ["john", "marry", "ben", "adam"]
print(stu)

stu.append("ramsay")    # 將'ramsay' 加到stu 的最後面
print(stu)

stu2 = ("dion", "carl", "john")    # 建立一個名為stu2 的序對
stu.extend(stu2)        # 將stu2 的資料加到stu 的最後面
print(stu)

stu.extend("test")        # 將"test"加到stu 後面
print(stu)

stu.remove("john")        # 找尋stu 裡面第一個"john"，並將它移除
print(stu)

print(stu.pop(2))        # 印出並刪除stu 中index 值為2 的元素
print(stu)

stu.insert(0, "dion")    # 在stu 的index 為0 的位置插入"dion"
print(stu)

print(stu.count("dion"))    # 計算"dion"在stu 當中出現幾次
stu.sort()                # 將stu 裡面的元素依Unicode 大小排序
print(stu)

stu.reverse()            # 反轉stu 裡面的元素順序
print(stu)

stu3 = stu                # 將stu 的物件參照指派給stu3
stu4 = stu.copy()        # 使用copy() 將stu 複製一份給stu4
print(stu == stu3)        # 使用is 跟== 運算子驗證stu、stu3、stu4 的關係
print(stu is stu3)
print(stu == stu4)
print(stu is stu4)
```

b. Output

```
['john', 'marry', 'ben', 'adam']
['john', 'marry', 'ben', 'adam', 'ramsay']
['john', 'marry', 'ben', 'adam', 'ramsay', 'dion', 'carl', 'john']
['john', 'marry', 'ben', 'adam', 'ramsay', 'dion', 'carl', 'john', 't', 'e', 's', 't']
['marry', 'ben', 'adam', 'ramsay', 'dion', 'carl', 'john', 't', 'e', 's', 't']
adam
['marry', 'ben', 'ramsay', 'dion', 'carl', 'john', 't', 'e', 's', 't']
```

```
['dion', 'marry', 'ben', 'ramsay', 'dion', 'carl', 'john',
't', 'e', 's', 't']
2
['ben', 'carl', 'dion', 'dion', 'e', 'john', 'marry',
'ramsay', 's', 't', 't']
['t', 't', 's', 'ramsay', 'marry', 'john', 'e', 'dion',
'dion', 'carl', 'ben']
True
True
True
False
```

3.1.3. Exercise

1. Book
    a. Question
    b. Answer
2. Classroom
    a. Question
    b. Answer

3.1.3.1. Book

3.1.3.1.1. Ex. 1

1. Question

> 請利用迴圈及串列的append和pop功能：
> 將「資料i (i=1, …, 10)」倒序放入串列folder中，例如(10,9,8...)。
> 然後，再將資料順序取出，例如(1,2,3...)。

2. Output

```
資料1
資料2
資料3
資料4
資料5
資料6
資料7
資料8
```

```
資料9
資料10
```

3.1.3.1.2. Ex. 2

1. Question

```
請將"The Beatles split up in April 1970."
用字串的split功能以半形空白分割為多個子字串，然後再印出index為2的子字串。
```

2. Output

```
split
```

3.1.3.1.3. Ex. 3

1. Question

```
若執行下列程式碼，則輸出值應為多少？

list2_3 = [[1, 2, 3], [4, 5, 6]]
print(list2_3[1][2])
```

3.1.3.1.4. Ex. 4

1. Question

```
讓使用者輸入一個2x3的矩陣，再翻轉後輸出。
```

3.1.3.1.5. Ex. 5

1. Question

```
請用程式計算並輸出兩矩陣的乘積。

 1 0 2        3 1
-1 3 1        2 1
            1 0
```

2. Output

```
5 1
4 2
```

3.1.3.1.6. Ex. 6

1. Question

> 請用程式檢視矩陣是否有反矩陣，
> 若無，請輸出`False`，
> 若有，請輸出該反矩陣。
>
> ```
> 2 5
> 3 7
> ```

2. Output

```
-7.0 5.0
3.0 -2.0
```

3.1.3.1.7. Ex. 7

1. Question

> 預設一串列`progression = [1, 2, 4, 8]`，
> 請設計一程式判斷該串列為等差數列或等比數列，並輸出下一項。

2. Output

> `[1, 2, 4, 8]` 為等比數列，下一項為 `16.0`

3.1.3.1.8. Ex. 8

1. Question

> 撰寫一程式讓使用者連續輸入數字，一次輸入一個。
> 每當使用者輸入一個數字時，程式便輸出統計至此數的眾數與該數的出現次數；
> 若眾數有兩個以上，則請用空白隔開。

Note

1. Start: 20170719
2. System Environment:

Listing 3.1.3.1.8.1 requirements.txt

```
sphinx==7.1.2                           # Sphinx
graphviz>=0.20.1                        # Graphviz
sphinxbootstrap4theme>=0.6.0            # Theme: Bootstrap
sphinx-material>=0.0.35                 # Theme: Material
sphinxcontrib-plantuml>=0.25            # PlantUML
sphinxcontrib.bibtex>=2.5.0             # Bibliography
sphinx-autorun>=1.1.1                   # ExecCode: pycon
sphinx-execute-code-python3>=0.3        # ExecCode
btd.sphinx.inheritance-diagram>=2.3.1   # Diagram
sphinx-copybutton>=0.5.1                # Copy button
sphinx_code_tabs>=0.5.3                 # Tabs
sphinx-immaterial>=0.11.3               # Tabs

#-----------------------------------------------------
#-- Library Upgrade Error by Library Itself
#   >> It needs to fix by library owner
#   >> After fixed, we need to try it later
#-----------------------------------------------------
pydantic==1.10.10                       # 2.0: sphinx compiler
error, 20230701

#-----------------------------------------------------
#-- Minor Extension
#-----------------------------------------------------
sphinxcontrib.httpdomain>=1.8.1         # HTTP API

#sphinxcontrib-blockdiag>=3.0.0         # Diagram: block
#sphinxcontrib-actdiag>=3.0.0           # Diagram: activity
#sphinxcontrib-nwdiag>=2.0.0            # Diagram: network
#sphinxcontrib-seqdiag>=3.0.0           # Diagram: sequence

#-----------------------------------------------------
#-- Still Wait For Upgrading Version
#-----------------------------------------------------

#-----------------------------------------------------
#-- Still Under Testing
#-----------------------------------------------------
#numpy>=1.24.2                          # Figure: numpy

#-----------------------------------------------------
#-- NOT Workable
#-----------------------------------------------------
#sphinxcontrib.jsdemo==0.1.4    # ExecCode: Need replace
add_js_file()
#jupyter-sphinx==0.4.0          # ExecCode: Need gcc compiler
```

```
#sphinxcontrib.slide==1.0.0     # Slide: Slideshare
#hieroglyph==2.1.0              # Slide: make slides
#matplotlib>=3.7.1              # Plot: Need Python >= v3.8
#manim==0.17.2                  # Diagram: scipy, numpy need
gcc
#sphinx_diagrams==0.4.0         # Diagram: Need GKE access
#sphinx-tabs>=3.4.1                     # Tabs: Conflict w/
sphinx-material
```