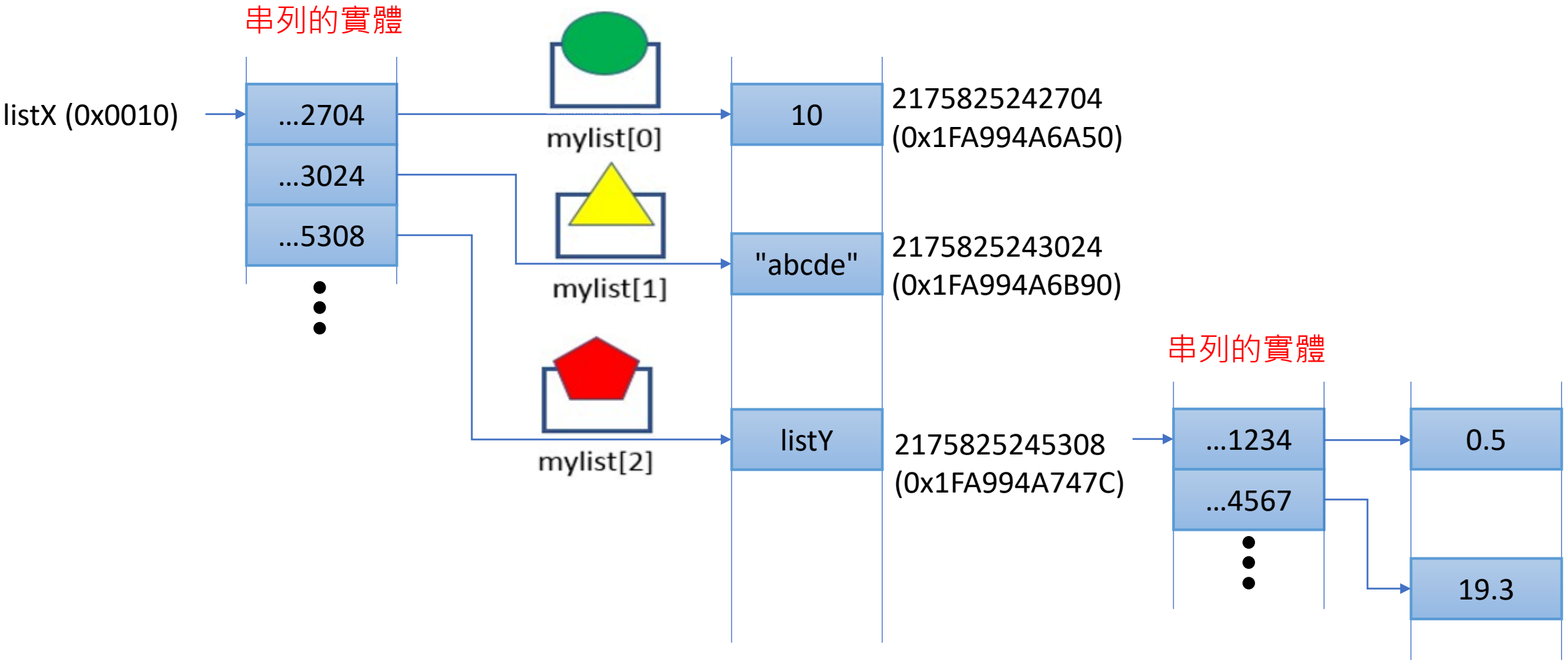
Chapter 6
串列

6.1 認識串列

- 串列 (list)
  - 由一系列元素組成的序列。
  - 資料內容可以修改。
  - 在在其他語言中（像是 C 系列的語言），相似的功能叫陣列(array)或鏈結串列(linked list)。
  - 串列也可以儲存不同資料型態。
  - 甚至一個串列也可以有其它串列、元組(tuple，第8章內容)或是字典(dict，第9章內容) ... 等當作是它的元素。

串列可以有不同元素, 可以用索引取得串列元素內容

6.1 認識串列


串列的實體
listX (0x0010)
...2704
...3024
...5308
mylist[0]
mylist[1]
mylist[2]
10
"abcde"
listY
2175825242704
(0x1FA994A6A50)
2175825243024
(0x1FA994A6B90)
2175825245308
(0x1FA994A747C)
串列的實體
...1234
...4567
0.5
19.3

6.1 認識串列

- mylist = [元素1, … , 元素n,]　　# mylist是假設的串列名稱
  - 串列中每一筆資料稱為元素，放在中括號[] 內，用","隔開。
  - 最後一筆資料元素n 右邊的逗點，可有可無。

```
1    myList = [1, 1, 2, 3, 5, 8, 13]
2    myListInList = [[1, 1, 2, 3, 5, 8, 13],
3                    ["上面是", "費波那契數"],
4                    ["再下一個元素是", 21],
5                    ]
```

6.1 認識串列

- mylist = [元素1, … , 元素n,]　　# mylist是假設的串列名稱

```
1   #水果攤賣的水果種類
2   fruits = ["Banana", "Apple", "Pineapple"]
3   print("水果種類：", fruits, "\n")
4
5   #NBA 球員 Lebron James 前五場比賽的得分
6   james = [23, 19, 22, 31, 18]
7   print(f"{james = }\n")
8
9   James = ["Lebron James", 23, 19, 22, 31, 18]
10  print(f"{James = }\n")
11
12  print(f"球員姓名：{James[0]}")
13  print(f"{James[0]} 第 1 場得分：{James[1]}")
14  print(f"{James[0]} 第 2 場得分：{James[2]}")
15  print(f"{James[0]} 第 3 場得分：{James[3]}")
16  print(f"{James[0]} 第 4 場得分：{James[4]}")
17  print(f"{James[0]} 第 5 場得分：{James[5]}")
```

```
水果種類： ['Banana', 'Apple', 'Pineapple']

james = [23, 19, 22, 31, 18]

James = ['Lebron James', 23, 19, 22, 31, 18]

球員姓名：Lebron James
Lebron James 第 1 場得分：23
Lebron James 第 2 場得分：19
Lebron James 第 3 場得分：22
Lebron James 第 4 場得分：31
Lebron James 第 5 場得分：18
```

6.1 認識串列

- mylist = [元素1, … , 元素n,]　　# mylist是假設的串列名稱
  - Index 可以用負的值
  - 底層會將 index 加上串列長度在處理。

```
1   #NBA 球員 Lebron James 前五場比賽的得分
2   James = ["Lebron James", 23, 19, 22, 31, 18]
3   print(f"{James = }\n")
4
5   print(f"球員姓名：{James[0]}")
6   print(f"{James[0]} 第 1 場得分：{James[-5]}") # -5 = 1-6
7   print(f"{James[0]} 第 2 場得分：{James[-4]}") # -4 = 2-6
8   print(f"{James[0]} 第 3 場得分：{James[-3]}") # -3 = 3-6
9   print(f"{James[0]} 第 4 場得分：{James[-2]}") # -2 = 4-6
10  print(f"{James[0]} 第 5 場得分：{James[-1]}") # -1 = 5-6
11  print(f"{James[0]} 第 6 場得分：{James[6]}")
```

-6 -5 -4 -3 -2 -1
0 1 2 3 4 5

```
球員姓名：Lebron James
Lebron James 第 1 場得分：23
Lebron James 第 2 場得分：19
Lebron James 第 3 場得分：22
Lebron James 第 4 場得分：31
Lebron James 第 5 場得分：18
Traceback (most recent call last):
  File "c:\[E]\PythonTest\test.py", line 11, in <module>
    print(f"{James[0]} 第 6 場得分：{James[6]}")
IndexError: list index out of range
```

6.1 認識串列

- mylist = [元素1, … , 元素n,]　　# mylist是假設的串列名稱
  - list unpacking

```
#NBA 球員 LeBron James 前五場比賽的得分
James = ["LeBron James", 23, 19, 22, 31, 18]
print(f"{James = }\n")

name = James[0]
score1 = James[1]
score2 = James[2]
score3 = James[3]
score4 = James[4]
score5 = James[5]
print(f"{name}前五場得分為 {score1}, {score2}, {score3}, {score4}, {score5}\n")

name, score1, score2, score3, score4, score5 = James
print(f"{name}前五場得分為 {score1}, {score2}, {score3}, {score4}, {score5}\n")
```

```
James = ['LeBron James', 23, 19, 22, 31, 18]

LeBron James前五場得分為 23, 19, 22, 31, 18

LeBron James前五場得分為 23, 19, 22, 31, 18
```

6.1 認識串列

- mylist = [元素1, … , 元素n,]　　# mylist是假設的串列名稱
  - 可以把字串用 list() 函數轉換成串列

```
1   string = "abcde"
2   listStr = list(string)
3   print(listStr)
4
5   int = 123
6   listInt = list(str(int))
7   print(listInt)
```

```
['a', 'b', 'c', 'd', 'e']
['1', '2', '3']
```

6.1 認識串列

- list slices: 取得串列的前幾個元素、後幾個元素、某個區間元素或是按照一定規則排列的元素，取出來後形成另一個新串列，也稱為子串列。
  - mylist[:] # 取得所有元素，形成兩個獨立串列
  - mylist[start:end] # 讀取從索引start到索引end-1 的串列元素
  - mylist[:end] # 取得串列最前面到end-1名
  - mylist[:-n] # 取得串列前面，不含最後n名
  - mylist[start:] # 取得串列索引start到最後
  - mylist[-n:] # 取得串列後n名
  - mylist[start:end:step] # 每隔step，讀取從索引start到(end-1)索引的串列

```
2  James = ["Lebron James", 23, 19, 22, 31, 18]
3  print(f"{id(James)}\n")
4  James1 = James
5  print(f"{id(James1)}\n")
6  James2 = James[:]
7  print(f"{id(James2)}\n")
```

```
2731822763200

2731822763200

2731823068736
```

6.1 認識串列

- list slices: 取得串列的前幾個元素、後幾個元素、某個區間元素或是按照一定規則排列的元素，取出來後形成另一個新串列，也稱為子串列。
  - mylist[:] # return all elements to form a new list
  - mylist[start:end] # index from start to <end
  - mylist[:end] # index from 0 to <end
  - mylist[:-n] # index from 0 to <length-n
  - mylist[start:] # index from start to <length
  - mylist[-n:] # index from length-n to <length
  - mylist[start:end:step] # index from start to <end with step
- 負號的 index，在 python 的底層實做上，會把負的 index 加上串列的長度，也就是 –n 會變成 –n+length。

6.1 認識串列

- list slices:

```
1   #NBA 球員 Lebron James 前五場比賽的得分
2   James = ["Lebron James", 23, 19, 22, 31, 18]
3   print(f"{James = }\n")
4
5   print(f"{James[1:5]}")          #index 1 to <5
6   print(f"{James[:5]}")           #index 0 to <5
7   print(f"{James[:-2]}")          #index 0 to <6-2
8   print(f"{James[1:]}")           #index 1 to <6
9   print(f"{James[-4:]}")          #index 6-4 to <6
10  print(f"{James[1:6:2]}")       #index 1 to <6, James[1::2]
11  print(f"{James[2:6:2]}")       #index 2 to <6, James[2::2]
12  print(f"{James[6:0:2]}")       #index 6 to <0
```

0 1 2 3 4 5

```
James = ['Lebron James', 23, 19, 22, 31, 18]

[23, 19, 22, 31]
['Lebron James', 23, 19, 22, 31]
['Lebron James', 23, 19, 22]
[23, 19, 22, 31, 18]
[19, 22, 31, 18]
[23, 22, 18]
[19, 31]
[]
```

*: step 用-1 會有什麼效果？

6.1 認識串列

- list slices:
  - step 是負的時候， start 要比 end 來的大。
  - 規則變成 index from start to >end

```
1   #NBA 球員 Lebron James 前五場比賽的得分
2   James = ["Lebron James", 23, 19, 22, 31, 18]
3   print(f"{James = }\n")
4
5   print(f"{James[::-1]}")          #reversed list
6   print(f"{James[0:6:-1]}")        #index 0 to >6, step -1
7   print(f"{James[6:0:-1]}")        #index 6 to >0, step -1
8   print(f"{James[2:-2:-1]}")       #index 2 to >6-2, step -1
9   print(f"{James[-2:2:-1]}")       #index 6-2 to >2, step -1
10  print(f"{James[5:-1:-1]}")       #index 5 to >6-1, step -1
11  print(f"{James[5:-6:-1]}")       #index 5 to >6-6, step -1
12  print(f"{James[-1:-7:-1]}")      #index 6-1 to >6-7, step -1
```

0 1 2 3 4 5

```
James = ['Lebron James', 23, 19, 22, 31, 18]

[18, 31, 22, 19, 23, 'Lebron James']
[]
[18, 31, 22, 19, 23]
[]
[31, 22]
[]
[18, 31, 22, 19, 23]
[18, 31, 22, 19, 23, 'Lebron James']
```

6.1 認識串列

- 若串列內容全部都為數值，則可使用 max(), min(), sum() 等函數獲得串列的最大,最小, 及加總值。
- 若串列內容全部都為字串，則可使用 max(), min()等函數獲得串列的 unicode 碼的最大最小值。（一個字元一個字元比較）

```
1   names = ["Lebron James", "Lebron Curry", "Durant"]
2   print(max(names))
3   print(min(names))
4
5   JamesScore = [23, 19, 22, 31, 18]
6   print(f"最高得分：{max(JamesScore)}")
7   print(f"最低得分：{min(JamesScore)}")
8   print(f"總得分：{sum(JamesScore)}")
```

```
Lebron James
Durant
最高得分：31
最低得分：18
總得分：113
```

6.1 認識串列

- 可用 len() 獲得串列中元素的個數。
- 修改串列中元素的值，可用串列名稱與索引值直接更改。

```
1   JamesScore = [23, 19, 22, 31, 18]
2   games = len(JamesScore)
3   print(f"經過 {games} 比賽，最高得分：{max(JamesScore)}")
4   print(f"經過 {games} 比賽，最低得分：{min(JamesScore)}")
5   print(f"經過 {games} 比賽，總得分：{sum(JamesScore)}")
6
7   JamesScore[3] = 40
8   print(f"{JamesScore}\n")
```

```
經過 5 比賽，最高得分：31
經過 5 比賽，最低得分：18
經過 5 比賽，總得分：113
[23, 19, 22, 40, 18]
```

6.1 認識串列

- 可以透過 + 或 + = 將串列結合。
- 將串列乘以一個數值，表示串列重複幾次。
- 刪除串列元素：（不知道實際上刪除了什麼內容）
  - del mylist[i] # 刪除index i的串列元素
  - del mylist[start:end] # 刪除從index start to <end-1 的串列元素
  - del mylist[start:end:step] # 每隔step，刪除從index start to <end-1的串列元素。

```
1   JamesScore = [23, 19, 22, 31, 18]
2   print(f"{JamesScore * 2}\n")
3   newGameScore = [20, 11]
4   JamesScore += newGameScore
5   print(f"{JamesScore}\n")
```

```
[23, 19, 22, 31, 18, 23, 19, 22, 31, 18]

[23, 19, 22, 31, 18, 20, 11]
```

6.1 認識串列

- 判斷串列是否是空串列：（利用長度==0）
- 刪除整個串列： del listName

```
1   JapanCars = ["Toyota", "Mazda", "Nissan"]
2   print(f"JapanCars 串列長度為 {len(JapanCars)}")
3   if len(JapanCars):  #符合 PEP 8， len(JapanCars) != 0，較不推薦
4       del JapanCars[0]
5       print(f"JapanCars 刪除元素 0 後串列長度為 {len(JapanCars)}")
6   else:
7       print("JapanCars 串列內沒有元素")
8
9   myList = []
10  print(f"myList 串列長度為 {len(myList)}")
11  if len(myList):
12      del myList[0]
13      print(f"myList 刪除元素 0 後串列長度為 {len(myList)}")
14  else:
15      print("myList 串列內沒有元素")
```

```
JapanCars 串列長度為 3
JapanCars 刪除元素 0 後串列長度為 2
myList 串列長度為 0
myList 串列內沒有元素
```

6.1 認識串列

- 補充多重指定與串列
  - 多重指定中，若左邊的變數較少，可用"*"變數的方式，將右邊多餘的內容用串列的方式打包給含"*"的變數。

```
a, b, *c = 1, 2, 3, 4, 5, 6
print(a, b, c)
a, *b, c = 1, 2, 3, 4, 5, 6
print(a, b, c)
```

```
1 2 [3, 4, 5, 6]
1 [2, 3, 4, 5] 6
```

6.2 Python 簡單的物件導向觀念

- 物件導向的程式設計(Object Oriented Programming)觀念裡，所有資料都是一個物件(Object)
  - 之前提到的所有變數型態，包含整數、浮點數、字串以及串列等等。
  - 物件會有方法(method)可以使用。
  - Python有為一些基本物件提供預設的方法供使用者使用。
  - 用法為：物件.方法()

6.2 Python 簡單的物件導向觀念

- 字串常用的方法：
  - lower( )：將字串轉成小寫字。
  - upper( )：將字串轉成大寫字。
  - title( )：將字串轉成第一個字母大寫，其它是小寫。
  - swapcase( )：將字串所有大寫改小寫，所有小寫改大寫。
  - rstrip( )：刪除字串尾端多餘的空白。
  - lstrip( )：刪除字串開始端多餘的空白。
  - strip( )：刪除字串頭尾兩邊多餘的空白。
  - center( )：字串在指定寬度置中對齊。
  - rjust( )：字串在指定寬度靠右對齊。
  - ljust( )：字串在指定寬度靠左對齊。
  - zfill( )：可以設定字串長度，原字串靠右對齊，左邊多餘空間補0。

6.2 Python 簡單的物件導向觀念

- 更改字串大小寫：lower( ) /upper( ) /title( )/swapcase( )

```
1   Ohtani = "OhtAnI ShOheI."
2   print(f"{Ohtani.upper()=}")
3   print(f"{Ohtani.lower()=}")
4   print(f"{Ohtani.title()=}")
5   print(f"{Ohtani.swapcase()=}")
```

```
Ohtani.upper()='OHTANI SHOHEI.'
Ohtani.lower()='ohtani shohei.'
Ohtani.title()='Ohtani Shohei.'
Ohtani.swapcase()='oHTaNi sHoHEi.'
```

- 刪除空白字元rstrip( )/lstrip( )/strip( )

```
1   Ohtani = " Ohtani Shohei "
2   print(f"{Ohtani.lstrip()=}")
3   print(f"{Ohtani.rstrip()=}")
4   print(f"{Ohtani.lstrip().rstrip()=}")
5   print(f"{Ohtani.strip()=}")
```

```
Ohtani.lstrip()='Ohtani Shohei '
Ohtani.rstrip()=' Ohtani Shohei'
Ohtani.lstrip().rstrip()='Ohtani Shohei'
Ohtani.strip()='Ohtani Shohei'
```

6.2 Python 簡單的物件導向觀念

- 格式化字串位置center( )/ljust( )/rjust( )/zfill( )

```
1   title = "Miin Wu School of Computing"
2   print(f"|{title.center(50)}|")
3   className = "Python Programing"
4   print(f"|{className.ljust(50)}|")
5   studentName = "Ohtani Shohei"
6   print(f"|{studentName.rjust(50)}|")
7   year = "2022"
8   print(f"|{year.zfill(50)}|")
```

```
|           Miin Wu School of Computing              |
|Python Programing                                   |
|                                        Ohtani Shohei|
|00000000000000000000000000000000000000000000002022|
```

6.2 Python 簡單的物件導向觀念

- 格式化字串位置center( )/ljust( )/rjust( )/zfill( )

```
1   title = "Miin Wu School of Computing"
2   print(f"|{title.center(50)}|")
3   className = "Python Programing"
4   print(f"|{className.ljust(50)}|")
5   studentName = "Ohtani Shohei"
6   print(f"|{studentName.rjust(50)}|")
7   year = "year2022"
8   print(f"|{year.zfill(50)}|")
9   year = "+2022"
10  print(f"|{year.zfill(50)}|")
11  year = "-2022"
12  print(f"|{year.zfill(50)}|")
13  year = "+-2022"
14  print(f"|{year.zfill(50)}|")
15  year = "+year2022"
16  print(f"|{year.zfill(50)}|")
```

```
|          Miin Wu School of Computing           |
|Python Programing                               |
|                                   Ohtani Shohei|
|000000000000000000000000000000000000year2022|
|+00000000000000000000000000000000000002022|
|-00000000000000000000000000000000000002022|
|+0000000000000000000000000000000000000-2022|
|+000000000000000000000000000000000year2022|
```

6.2 Python 簡單的物件導向觀念

- 字串格式判斷 islower()/isupper()/isdigit()/isalpha()
  - 要全部都是小寫/大寫/數字/英文字 結果才會是 True

```
1   str1 = "aabbcc"
2   print(str1.isupper())
3   print(str1.islower())
4   print(str1.isdigit())
5   str2 = "123"
6   print(str2.isdigit())
7   print(str1.isalpha())
8   print(str2.isalpha())
9   str3 = "AAbbcc"
10  print(str3.isupper())
11  print(str3.islower())
```

```
False
True
False
True
True
False
False
False
```

6.2 Python 簡單的物件導向觀念

- dir獲得系統內建物件的方法
  - 以字串為例：

```
>>> string = "abc"
>>> dir(string)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq_
_', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__
gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__',
'__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subcl
asshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expa
ndtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', '
isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspa
ce', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partiti
on', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartit
ion', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
```

dir(str)

  - 用 help 獲得方法的說明。

```
>>> help(string.islower)
Help on built-in function islower:

islower() method of builtins.str instance
    Return True if the string is a lowercase string, False otherwise.

    A string is lowercase if all cased characters in the string are lowercase and
    there is at least one cased character in the string.
```

help(str.islower)

6.3 串列的方法

- dir獲得串列的方法

```
>>> dir([])
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__
', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash
__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt
__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__r
mul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear'
, 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

6.4 增加與刪除串列元素

- append( )：在串列末端增加元素

```
1   JapanCars = ["Toyota", "Mazda", "Nissan"]
2   print(f"{JapanCars}")
3   JapanCars[3] = "Lexus"
```

```
['Toyota', 'Mazda', 'Nissan']
Traceback (most recent call last):
  File "e:\PythonTest\test.py", line 3, in <module>
    JapanCars[3] = "Lexus"
IndexError: list assignment index out of range
```

用 + 或 += 實現新增元素？

```
1   JapanCars = ["Toyota", "Mazda", "Nissan"]
2   print(f"{JapanCars}")
3   JapanCars.append("Lexus")
4   print(f"{JapanCars}")
```

```
['Toyota', 'Mazda', 'Nissan']
['Toyota', 'Mazda', 'Nissan', 'Lexus']
```

- 預留串列空間

```
1   x = [None] * 3          #宣告長度為3，不指定元素形式的串列
2   print(x, len(x))
3   x[0] = 1
4   x[1] = 2
5   x[2] = "12"
6   print(x)
```

```
[None, None, None] 3
[1, 2, '12']
```

6.4 增加與刪除串列元素

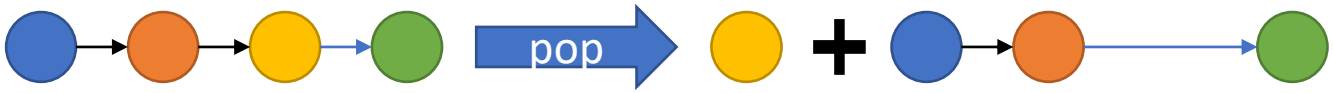
- insert(index, value)：在 index之前插入串列元素 value。

```
1   JapanCars = ["Toyota", "Mazda", "Nissan"]
2   print(f"{JapanCars}")
3   JapanCars.insert(1, "Lexus")
4   print(f"{JapanCars}")
5   JapanCars.insert(0, "Infinity")
6   print(f"{JapanCars}")
7   JapanCars.insert(-1, "Honda")
8   print(f"{JapanCars}")
```

```
['Toyota', 'Mazda', 'Nissan']
['Toyota', 'Lexus', 'Mazda', 'Nissan']
['Infinity', 'Toyota', 'Lexus', 'Mazda', 'Nissan']
['Infinity', 'Toyota', 'Lexus', 'Mazda', 'Honda', 'Nissan']
```

6.4 增加與刪除串列元素

- pop()：將串列的最後一個元素取出，並將串列中最後一個元素刪除。
- pop(index)：將串列中索引index的元素取出，並將串列中索引index的元素刪除。




```
1   JapanCars = ["Infinity", "Toyota", "Lexus", "Mazda", "Honda", "Nissan"]
2   print(f"{JapanCars}")
3   lastBrand = JapanCars.pop()
4   print(f"{lastBrand} + {JapanCars}")
5   brand3 = JapanCars.pop(3)
6   print(f"{brand3} + {lastBrand} + {JapanCars}")
7   brand4 = JapanCars.pop(4)
8   print(f"{JapanCars}")
```

```
['Infinity', 'Toyota', 'Lexus', 'Mazda', 'Honda', 'Nissan']
Nissan + ['Infinity', 'Toyota', 'Lexus', 'Mazda', 'Honda']
Mazda + Nissan + ['Infinity', 'Toyota', 'Lexus', 'Honda']
Traceback (most recent call last):
  File "e:\PythonTest\test.py", line 7, in <module>
    brand4 = JapanCars.pop(4)
IndexError: pop index out of range
```

6.4 增加與刪除串列元素

- remove(value)：將串列中元素值為 value 的元素刪除。

```
1   JapanCars = ["Infinity", "Toyota", "Lexus", "Mazda", "Honda", "Nissan", "Toyota"]
2   print(f"{JapanCars}")
3   JapanCars.remove("Lexus")
4   print(f"After remode Lexus: {JapanCars}")
5   JapanCars.remove("Toyota")
6   print(f"After remode Toyota: {JapanCars}") *:想將相同的元素移除乾淨要用迴圈
7   JapanCars.remove("Lexus")
8   print(f"After remode Lexus: {JapanCars}")
```

```
['Infinity', 'Toyota', 'Lexus', 'Mazda', 'Honda', 'Nissan', 'Toyota']
After remode Lexus: ['Infinity', 'Toyota', 'Mazda', 'Honda', 'Nissan', 'Toyota']
After remode Toyota: ['Infinity', 'Mazda', 'Honda', 'Nissan', 'Toyota']
Traceback (most recent call last):
  File "e:\PythonTest\test.py", line 7, in <module>
    JapanCars.remove("Lexus")
ValueError: list.remove(x): x not in list
```

6.5 串列的排序

- reverse()：將串列中元素值顛倒排序。

```
1   JapanCars = ["Infinity", "Toyota", "Lexus", "Mazda", "Honda", "Nissan", "Toyota"]
2   print(f"{JapanCars}")
3   print(f"print list in reversed order: {JapanCars[::-1]}")
4   JapanCars.reverse()     #reverse list
5   print(f"{JapanCars}")
```

```
['Infinity', 'Toyota', 'Lexus', 'Mazda', 'Honda', 'Nissan', 'Toyota']
print list in reversed order: ['Toyota', 'Nissan', 'Honda', 'Mazda', 'Lexus', 'Toyota', 'Infinity']
['Toyota', 'Nissan', 'Honda', 'Mazda', 'Lexus', 'Toyota', 'Infinity']
```

6.5 串列的排序

- sort()：將串列中元素值由小到大排序。

```
1   JapanCars = ["Infinity", "Toyota", "Lexus", "Mazda", "Honda", "Nissan", "Toyota"]
2   print(f"{JapanCars}")
3   JapanCars.sort()
4   print(f"{JapanCars}")
5   JapanCars.sort(reverse = True)
6   print(f"{JapanCars}")
```

```
['Infinity', 'Toyota', 'Lexus', 'Mazda', 'Honda', 'Nissan', 'Toyota']
['Honda', 'Infinity', 'Lexus', 'Mazda', 'Nissan', 'Toyota', 'Toyota']
['Toyota', 'Toyota', 'Nissan', 'Mazda', 'Lexus', 'Infinity', 'Honda']
```

6.5 串列的排序

- 函數 sorted(list, reverse=False)：將串列排序，產生新串列。
  - reverse= False(由小到大) / True(由大到小)

```
1   JapanCars = ["Infinity", "Toyota", "Lexus", "Mazda", "Honda", "Nissan"]
2   print(f"{JapanCars}")
3   newList = sorted(JapanCars)
4   print(f"{JapanCars=}")
5   print(f"{newList=}")
6   newList = sorted(JapanCars, reverse=True)
7   print(f"{newList=}")
```

```
['Infinity', 'Toyota', 'Lexus', 'Mazda', 'Honda', 'Nissan']
JapanCars=['Infinity', 'Toyota', 'Lexus', 'Mazda', 'Honda', 'Nissan']
newList=['Honda', 'Infinity', 'Lexus', 'Mazda', 'Nissan', 'Toyota']
newList=['Toyota', 'Nissan', 'Mazda', 'Lexus', 'Infinity', 'Honda']
```

6.6 串列的進階操作

- index(value)：傳回串列中元素值 value 第一次出現的索引值。

```
1   JapanCars = ["Infinity", "Toyota", "Lexus", "Mazda", "Honda", "Nissan", "Toyota"]
2   print(f"{JapanCars}")
3   idxLexus = JapanCars.index("Lexus")
4   print(f"Index of Lexus in JapanCars is {idxLexus}")
5   idxToyota = JapanCars.index("Toyota")
6   print(f"Index of Toyota in JapanCars is {idxToyota}")
7   idxMazda = JapanCars.index("mazda")
8   print(f"Index of mazda in JapanCars is {idxMazda}")
```

```
['Infinity', 'Toyota', 'Lexus', 'Mazda', 'Honda', 'Nissan', 'Toyota']
Index of Lexus in JapanCars is 2
Index of Toyota in JapanCars is 1
Traceback (most recent call last):
  File "e:\PythonTest\test.py", line 7, in <module>
    idxMazda = JapanCars.index("mazda")
ValueError: 'mazda' is not in list
```

6.6 串列的進階操作

- index(value)：傳回串列中元素值 value 第一次出現的索引值。

```
1   JapanCars = ["Infinity", "Toyota", "Lexus", "Mazda", "Honda", "Nissan", "Toyota"]
2   print(f"{JapanCars}")
3   idxLexus = JapanCars.index("Lexus")
4   print(f"Index of Lexus in JapanCars is {idxLexus}")
5   idxToyota = JapanCars.index("Toyota")
6   print(f"Index of Toyota in JapanCars is {idxToyota}")
7   #idxMazda = JapanCars.index("mazda")
8   #print(f"Index of mazda in JapanCars is {idxMazda}")
9
10  newJPCars = [None] * len(JapanCars)
11  newJPCars[0] = JapanCars[0].lower()
12  newJPCars[1] = JapanCars[1].lower()
13  newJPCars[2] = JapanCars[2].lower()
14  newJPCars[3] = JapanCars[3].lower()
15  newJPCars[4] = JapanCars[4].lower()
16  newJPCars[5] = JapanCars[5].lower()
17  newJPCars[6] = JapanCars[6].lower()
18  print(newJPCars)
19  idxMazda = newJPCars.index("mazda")
20  print(f"Index of mazda in JapanCars is {idxMazda}")
```

```
['Infinity', 'Toyota', 'Lexus', 'Mazda', 'Honda', 'Nissan', 'Toyota']
Index of Lexus in JapanCars is 2
Index of Toyota in JapanCars is 1
['infinity', 'toyota', 'lexus', 'mazda', 'honda', 'nissan', 'toyota']
Index of mazda in JapanCars is 3
```

6.6 串列的進階操作

- count(value)：傳回串列中元素值 value 出現的次數。

```
1   James = ["LeBron James", 23, 22, 22, 31, 18]
2   count23 = James.count(23)
3   count22 = James.count(22)
4   count50 = James.count(50)
5   print(f"{James[0]} {len(James)-1} 場比賽，得23分的次數為{count23}")
6   print(f"{James[0]} {len(James)-1} 場比賽，得22分的次數為{count22}")
7   print(f"{James[0]} {len(James)-1} 場比賽，得50分的次數為{count50}")
```

```
LeBron James 5 場比賽，得23分的次數為1
LeBron James 5 場比賽，得22分的次數為2
LeBron James 5 場比賽，得50分的次數為0
```

6.7 串列內含串列(二維串列)

- num = [1, 2, 3, 4, 5, [6, 7, 8]]

```
1   num = [1, 2, 3, 4, 5, [6, 7, 8]]
2   print(num[5])
3   print(type(num[5]))
4   print(num[5][0], num[5][1], num[5][2])
```

```
[6, 7, 8]
<class 'list'>
6 7 8
```

```
1   James = [["LeBron James", "SF", "12/30/84"], 23, 22, 22, 31, 18]
2   games = len(James)
3   scoreMax = max(James[1:games])
4   idx = James.index(scoreMax)
5   #name = James[0][0]
6   #position = James[0][1]
7   #birthDay = James[0][2]
8   name, position, birthDay = James[0]    #多重指定
9   print((f"名字\t: {name}\n"
10          f"位置\t: {position}\n"
11          f"生日\t: {birthDay}\n"
12          f"在第 {idx} 場比賽得最高分 {scoreMax}")
13  )
```

```
名字    : LeBron James
位置    : SF
生日    : 12/30/84
在第 4 場比賽得最高分 31
```

6.7 串列內含串列(二維串列)

- append(list): 除了 append 元素外，也可 append 串列，形成串列內含串列。

```
1   JapanCars = ["Infinity", "Toyota", "Lexus", "Mazda"]
2   JapanCars1 = ["Honda", "Nissan"]
3   JapanCars.append(JapanCars1)
4   print(f"{JapanCars}")
```

```
['Infinity', 'Toyota', 'Lexus', 'Mazda', ['Honda', 'Nissan']]
```

- extend(list): 跟 append 類似，但只能用於兩個串列。同時會將 list 分解成元素，一一 append 到串列中。(+=)

```
1   JapanCars = ["Infinity", "Toyota", "Lexus", "Mazda"]
2   JapanCars1 = ["Honda", "Nissan"]
3   JapanCars.extend(JapanCars1)
4   print(f"{JapanCars}")
```

```
['Infinity', 'Toyota', 'Lexus', 'Mazda', 'Honda', 'Nissan']
```

6.7 串列內含串列(二維串列)

姓名	國文	英文	數學	總分
王小明	80	95	88	0
蘇小花	98	97	96	0
謝大樹	94	93	85	0
李大呆	91	94	95	0
陳小比	92	97	90	0

姓名	國文	英文	數學	總分
[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
[3][0]	[3][1]	[3][2]	[3][3]	[3][4]
[4][0]	[4][1]	[4][2]	[4][3]	[4][4]

```
1   scoreTable = [["王小明", 80, 95, 88, 0],
2                 ["蘇小花", 98, 97, 96, 0],
3                 ["謝大樹", 94, 93, 85, 0],
4                 ["李大呆", 91, 94, 95, 0],
5                 ["陳小比", 92, 97, 90, 0],
6                ]
7
8   scoreTable[0][4] = sum(scoreTable[0][1:4])
9   scoreTable[1][4] = sum(scoreTable[1][1:4])
10  scoreTable[2][4] = sum(scoreTable[2][1:4])
11  scoreTable[3][4] = sum(scoreTable[3][1:4])
12  scoreTable[4][4] = sum(scoreTable[4][1:4])
13
14  print(f"  姓名  , 國文, 英文, 數學, 總分")
15  print(f"{scoreTable[0]}")
16  print(f"{scoreTable[1]}")
17  print(f"{scoreTable[2]}")
18  print(f"{scoreTable[3]}")
19  print(f"{scoreTable[4]}")
```

```
  姓名  , 國文, 英文, 數學, 總分
['王小明', 80, 95, 88, 263]
['蘇小花', 98, 97, 96, 291]
['謝大樹', 94, 93, 85, 272]
['李大呆', 91, 94, 95, 280]
['陳小比', 92, 97, 90, 279]
```

6.8串列的賦值與切片拷貝

- 串列的賦值

```
1   mySports = ["baseball", "basketball"]
2   friendSports = mySports
3   print(f"我喜歡的運動\t：{mySports}")
4   print(f"朋友喜歡的運動\t：{friendSports}")
5
6   mySports.append("soccer")
7   friendSports.append("badminton")
8   print(f"我喜歡的運動\t：{mySports}")
9   print(f"朋友喜歡的運動\t：{friendSports}")
```

```
我喜歡的運動    ：['baseball', 'basketball']
朋友喜歡的運動   ：['baseball', 'basketball']
我喜歡的運動    ：['baseball', 'basketball', 'soccer', 'badminton']
朋友喜歡的運動   ：['baseball', 'basketball', 'soccer', 'badminton']
```

6.8串列的賦值與切片拷貝

- 串列的位址(address)

```
1   mySports = ["baseball", "basketball"]
2   friendSports = mySports
3   print(f"id(mySports)\t\t={id(mySports)}\nid(friendSports)\t={id(friendSports)}")
4   print(f"我喜歡的運動\t：{mySports}")
5   print(f"朋友喜歡的運動\t：{friendSports}")
6
7   mySports.append("soccer")
8   friendSports.append("badminton")
9   print(f"我喜歡的運動\t：{mySports}")
10  print(f"朋友喜歡的運動\t：{friendSports}")
```

```
id(mySports)            =3060112641344
id(friendSports)        =3060112641344
我喜歡的運動    ：['baseball', 'basketball']
朋友喜歡的運動  ：['baseball', 'basketball']
我喜歡的運動    ：['baseball', 'basketball', 'soccer', 'badminton']
朋友喜歡的運動  ：['baseball', 'basketball', 'soccer', 'badminton']
```

- 等號僅僅將變數位址等過去，而非將內容拷貝過去。

6.8串列的賦值與切片拷貝

- 串列的切片(slice)

```
1   mySports = ["baseball", "basketball"]
2   friendSports = mySports[:]
3   print(f"id(mySports)\t\t={id(mySports)}\nid(friendSports)\t={id(friendSports)}")
4   print(f"我喜歡的運動\t：{mySports}")
5   print(f"朋友喜歡的運動\t：{friendSports}")
6
7   mySports.append("soccer")
8   friendSports.append("badminton")
9   print(f"我喜歡的運動\t：{mySports}")
10  print(f"朋友喜歡的運動\t：{friendSports}")
```

```
id(mySports)            =2847601413376
id(friendSports)        =2847601718976
我喜歡的運動    ：['baseball', 'basketball']
朋友喜歡的運動  ：['baseball', 'basketball']
我喜歡的運動    ：['baseball', 'basketball', 'soccer']
朋友喜歡的運動  ：['baseball', 'basketball', 'badminton']
```

  - slice 會產生一個新的串列，此時 friendSports 會指向新的串列的位址。

6.8串列的賦值與切片拷貝

- 串列的拷貝(copy)與深拷貝(deepcopy)

```
1   listA = [1, 2, 3, [4, 5, 6]]
2   listB = listA.copy()
3   print(f"id(listA)\t={id(listA)}\nid(listB)\t={id(listB)}")
4   print((f"{listA=}\n"
5          f"{listB=}\n"))
6
7   listA.append(7)
8   print((f"{listA=}\n"
9          f"{listB=}\n"))
10
11  listA[3].append(8)
12  print((f"{listA=}\n"
13         f"{listB=}\n"))
14
15  print(f"id(listA[3])\t={id(listA[3])}\nid(listB[3])\t={id(listB[3])}")
```

```
id(listA)       =2745925498624
id(listB)       =2745925543616
listA=[1, 2, 3, [4, 5, 6]]
listB=[1, 2, 3, [4, 5, 6]]

listA=[1, 2, 3, [4, 5, 6], 7]
listB=[1, 2, 3, [4, 5, 6]]

listA=[1, 2, 3, [4, 5, 6, 8], 7]
listB=[1, 2, 3, [4, 5, 6, 8]]

id(listA[3])    =2745925193024
id(listB[3])    =2745925193024
```

6.8串列的賦值與切片拷貝

- 串列的拷貝(copy)與深拷貝(deepcopy)

```
1   import copy
2   listA = [1, 2, 3, [4, 5, 6]]
3   listB = copy.deepcopy(listA)
4   print(f"id(listA)\t={id(listA)}\nid(listB)\t={id(listB)}")
5   print(f"id(listA[3])\t={id(listA[3])}\nid(listB[3])\t={id(listB[3])}")
6
7   print((f"{listA=}\n"
8          f"{listB=}\n"))
9
10  listA.append(7)
11
12  listA[3].append(8)
13  print((f"{listA=}\n"
14         f"{listB=}\n"))
```

```
id(listA)       =1663302281600
id(listB)       =1663302281536
id(listA[3])    =1663302841600
id(listB[3])    =1663302840448
listA=[1, 2, 3, [4, 5, 6]]
listB=[1, 2, 3, [4, 5, 6]]

listA=[1, 2, 3, [4, 5, 6, 8], 7]
listB=[1, 2, 3, [4, 5, 6]]
```

6.9 再談字串

- 字串可想像為一個由字元組成的序列。字串與串列不同的是，字串內的單一元素(字元)內容是不可更改的。
- 字串的索引：

```
1   string = "Python"
2   string[2] = "a"
```

```
Traceback (most recent call last):
  File "e:\PythonTest\test.py", line 2, in <module>
    string[2] = "a"
TypeError: 'str' object does not support item assignment
```

```
1   string = "Python"
2   print(f"{string[0] =}",
3         f"\n{string[1] =}",
4         f"\n{string[2] =}",
5         f"\n{string[3] =}",
6         f"\n{string[4] =}",
7         f"\n{string[5] =}")
8
9   print(f"{string[-1] =}",
10        f"\n{string[-2] =}",
11        f"\n{string[-3] =}",
12        f"\n{string[-4] =}",
13        f"\n{string[-5] =}",
14        f"\n{string[-6] =}")
15
16  c1, c2, c3, c4 ,c5, c6 = string
17  print("多重指定的輸出：", c1, c2, c3, c4 ,c5, c6)
```

```
string[0] ='P'
string[1] ='y'
string[2] ='t'
string[3] ='h'
string[4] ='o'
string[5] ='n'
string[-1] ='n'
string[-2] ='o'
string[-3] ='h'
string[-4] ='t'
string[-5] ='y'
string[-6] ='P'
多重指定的輸出： P y t h o n
```

6.9 再談字串

- 字串的切片：

```
1   string = "Deep Learning"
2   print(f"字串的第 0-2 個元素: {string[0:3]}")
3   print(f"字串的第 1-3 個元素: {string[1:4]}")
4   print(f"字串的第 1, 3, 5 個元素: {string[1:6:2]}")
5   print(f"字串的第 1 到最後的元素: {string[1:]}")
6   print(f"字串的最後 3 個元素: {string[-3:]}")
7   print(f"字串的reverse: {string[::-1]}")
```

```
字串的第 0-2 個元素: Dee
字串的第 1-3 個元素: eep
字串的第 1, 3, 5 個元素: epL
字串的第 1 到最後的元素: eep Learning
字串的最後 3 個元素: ing
字串的reverse: gninraeL peeD
```

- len(), max(), min()

```
1   string = "DeepLearning"
2   print(f"字串長度: {len(string)}")
3   max_ch = max(string)
4   min_ch = min(string)
5   print(f"最大的字元: {max_ch}, unicode 為 {ord(max_ch)}")
6   print(f"最小的字元: {min_ch}, unicode 為 {ord(min_ch)}")
```

```
字串長度: 12
最大的字元: r, unicode 為 114
最小的字元: D, unicode 為 68
```

6.9 再談字串

- 字串的split()：以空格或其他符號為分隔符號，將字串拆開成一個串列。
- 串列的 join()：使用 join 將串列組合成一個字串。

```
1   string1 = "Deep Learning"
2   string2 = "E:\\Python\\test.py" # r"E:\Python\test.py"
3   strList1 = string1.split( )
4   strList2 = string2.split("\\")
5   print(f"{strList1 = }")
6   print(f"{strList2 = }")
7
8   ch1 = "-"
9   strDL = ch1.join(strList1)
10  print(strDL)
11  ch2 = "/"
12  strPath = ch2.join(strList2)
13  print(strPath)
```

```
strList1 = ['Deep', 'Learning']
strList2 = ['E:', 'Python', 'test.py']
Deep-Learning
E:/Python/test.py
```

6.9 再談字串

- 子字串搜尋與索引：
  - find( ): 從頭尋找子字串，並回傳第一次出現的索引值。若找不到則回傳-1。
  - rfind( )：從尾尋找子字串，並回傳第一次出現的索引值。若找不到則回傳-1。
  - index( )：從頭尋找子字串，並回傳第一次出現的索引值。若找不到則產生例外錯誤。
  - rindex( )：從尾尋找子字串，並回傳第一次出現的索引值。若找不到則產生例外錯誤。
  - count( )：列出子字串出現次數。
  - isalnum( )：判斷字串是否只有數字或字母，沒有特殊符號(空格, ",", ...)。

```
1   str1 = "Deep Learning Means Learning Deeply."
2   subStr = "Learn"
3   print(str1.count(subStr))
4   print(str1.find(subStr))
5   print(str1.rfind(subStr))
6   print(str1.find("deep"))
7   print(str1.index("deep"))
```

```
2
5
20
-1
Traceback (most recent call last):
  File "e:\PythonTest\test.py", line 7, in <module>
    print(str1.index("deep"))
ValueError: substring not found
```

6.9 再談字串

- 字串的其它方法：
  - startswith( )：可以列出字串啟始文字是否是特定子字串。
  - endswith( )：可以列出字串結束文字是否是特定子字串。
  - replace(ch1,ch2)：將ch1字串用ch2字串取代。

```
1   msg = "Avenger Tony told Avenger Steve that stones are given to Avenger Thor"
2
3   print(f"String starts with Avenger is {msg.startswith('Avenger')}.")
4   print(f"String ends with Avenger is {msg.endswith('Avenger')}.")
5   print(f"Avenger shows up {msg.count('Avenger')} times.")
6   msg1 = msg.replace("Steve", "Strange")
7   print(f"New string: {msg1}")
```

```
String starts with Avenger is True.
String ends with Avenger is False.
Avenger shows up 3 times.
New string: Avenger Tony told Avenger Strange that stones are given to Avenger Thor
```

6.10 in 和 not in 運算式

- 用於判斷一個物件是否屬於另外一個物件。物件可以是字串(string)、串列(list)、元組(Tuple)、字典(Dict)。
  - boolValue = obj1 in obj2
  - boolValue = obj1 not in obj2

```
1   password = "SpyderMan"
2   ch = input("Input a character:")
3   if ch in password:
4       print(f"{ch} is in the password.")
5   else:
6       print(f"{ch} is not in the password.")
7
8   if ch not in password:
9       print(f"{ch} is not in the password.")
10  else:
11      print(f"{ch} is in the password.")
```

```
Input a character:a
a is in the password.
a is in the password.
```

```
Input a character:c
c is not in the password.
c is not in the password.
```

6.10 in 和 not in 運算式

- 用於判斷一個物件是否屬於另外一個物件。物件可以是字串(string)、串列(list)、元組(Tuple)、字典(Dict)。

```
1   fruits = ["Banana", "Apple", "Watermelon"]
2   oneFruit = input("Please input the fruit name:")
3   if oneFruit in fruits:
4       print("We have already have this fruit.")
5   else:
6       fruits.append(oneFruit)
7       print("We have added the fruit into the list."
8       f"\n{fruits=}")
```

```
Please input the fruit name:Lychee
We have added the fruit into the list.
fruits=['Banana', 'Apple', 'Watermelon', 'Lychee']
```

6.11 is 或 is not 運算式

- 用於判斷兩個物件是否相同。
  - 不是指內容相同，而物件變數是否指向同一個記憶體位址。
  - 物件可以是變數、字串(string)、串列(list)、元組(Tuple)、字典(Dict)。

boolValue = obj1 is obj2
boolValue = obj1 is not obj2

```
1   x = 10
2   y = 10
3   z = 15
4   w = 20
5   print(f"value: {x = }, {y = }, {z = }, {w = }")
6   print(f"address: {id(x) = }, {id(y) = }, {id(z) = }, {id(w) = }")
7
8   w = x    #try w = 10, what will happen
9   print(f"value: {x = }, {y = }, {z = }, {w = }")
10  print(f"address: {id(x) = }, {id(y) = }, {id(z) = }, {id(w) = }")
```

```
value: x = 10, y = 10, z = 15, w = 20
address: id(x) = 1922638965328, id(y) = 1922638965328, id(z) = 1922638965488, id(w) = 1922638965648
value: x = 10, y = 10, z = 15, w = 10
address: id(x) = 1922638965328, id(y) = 1922638965328, id(z) = 1922638965488, id(w) = 1922638965328
```

6.11 is 或 is not 運算式

```
 1  x = 10
 2  y = 10
 3  z = 15
 4  w = z-5
 5  bool_XIsY = x is y
 6  print(f"{id(x) = }, {id(y) = }, {bool_XIsY = }")
 7  bool_XIsZ = x is z
 8  print(f"{id(x) = }, {id(z) = }, {bool_XIsZ = }")
 9  bool_XIsW = x is w
10  print(f"{id(x) = }, {id(w) = }, {bool_XIsW = }")
11  bool_XIsNotY = x is not y
12  print(f"{id(x) = }, {id(y) = }, {bool_XIsNotY = }")
13  bool_XIsNotZ = x is not z
14  print(f"{id(x) = }, {id(z) = }, {bool_XIsNotZ = }")
15  bool_XIsNotW = x is not w
16  print(f"{id(x) = }, {id(w) = }, {bool_XIsNotW = }")
```

```
id(x) = 2324982164048, id(y) = 2324982164048, bool_XIsY = True
id(x) = 2324982164048, id(z) = 2324982164208, bool_XIsZ = False
id(x) = 2324982164048, id(w) = 2324982164048, bool_XIsW = True
id(x) = 2324982164048, id(y) = 2324982164048, bool_XIsNotY = False
id(x) = 2324982164048, id(z) = 2324982164208, bool_XIsNotZ = True
id(x) = 2324982164048, id(w) = 2324982164048, bool_XIsNotW = False
```

6.11 is 或 is not 運算式

```
 1  mySports = ["baseball", "basketball"]
 2  sports1 = mySports
 3  sports2 = mySports[:]
 4  print(f"{mySports = }, id(mySports)\t= {id(mySports)}")
 5  print(f"{sports1  = }, id(mySports1)\t= {id(sports1)}")
 6  print(f"{sports2  = }, id(mySports2)\t= {id(sports2)}")
 7  boolVal1 = mySports is sports1
 8  boolVal2 = mySports is sports2
 9  boolValNot1 = mySports is not sports1
10  boolValNot2 = mySports is not sports2
11  print(f"mySports is sports1:\t\t{boolVal1}")
12  print(f"mySports is sports2:\t\t{boolVal2}")
13  print(f"mySports is not sports1:\t{boolValNot1}")
14  print(f"mySports is not sports2:\t{boolValNot2}")
```

```
mySports = ['baseball', 'basketball'], id(mySports)     = 1883396263232
sports1  = ['baseball', 'basketball'], id(mySports1)    = 1883396263232
sports2  = ['baseball', 'basketball'], id(mySports2)    = 1883396568832
mySports is sports1:            True
mySports is sports2:            False
mySports is not sports1:        False
mySports is not sports2:        True
```

6.11 is 或 is not 運算式

- None 資料型態不是空字串、空串列。

```
x = []
if (x is None):
    print("x is None")
else:
    print("x is not None")
```

```
x is not None
```

6.12 enumerate 物件

- enumerate() 函數可以將 iterable （可迭代）類數值的元素用索引值與元素的配對方式回傳。
  - 回傳的數據稱為 enumerate 物件。
  - 為可迭代的物件增加索引值。
  - iterable 類數值可以是串列(list)、元組(Tuple)、集合(Set)...等。

  obj = enumerate(iterable [, start = 0])
    - 若省略 start = 的設定，預設索引值為 0。

```
1   drinks = ["tea", "coffee", "milk"]
2   enum_drinks = enumerate(drinks)
3   print(f"{enum_drinks}")
4   print(f"{type(enum_drinks)=}")
5   print(f"{list(enum_drinks)=}")
6
7   enum_drinks10 = enumerate(drinks, 10)
8   print(f"{list(enum_drinks10)=}")
```

```
<enumerate object at 0x00000182D8B2A940>
type(enum_drinks)=<class 'enumerate'>
list(enum_drinks)=[(0, 'tea'), (1, 'coffee'), (2, 'milk')]
list(enum_drinks10)=[(10, 'tea'), (11, 'coffee'), (12, 'milk')]
```

6.13 動手練習

- 帳號管理系統：1. 事先建立帳號串列。2. 要求輸入一個帳號，若是存在帳號串列中則輸出"歡迎登入"，若是不存在帳號串列中則輸出"帳號錯誤"。

- 文件加密：建立ABC ... Z字母的字串，然後使用切片取得前n個英文字母，與後26-n個英文字母。最後組合，可以得到新的字母排序。

n=3