Title: ตำราวิชา Problem Solving and Computer Programming - PSCP Book

Author: รศ.ดร. โชติพัชร์ ภรณวลัย

Language: th-TH

Date: 17 กันยายน 2566

Ref: Think Python Chapter 10, 12

Chapter 6: List and Tuple

List

List เป็นข้อมูลแบบมีลำดับ ลักษณะเดียวกับ String โดยมีการใช้ index อ้างถึง element หรือ item ที่อยู่ข้างใน โดย index จะเริ่มจาก 0 เช่นเดียวกับ String และตัวสุดท้ายจะมี index เป็น -1

แต่สิ่งที่แตกต่างระหว่าง String กับ List คือ item ใน String จะเป็นตัวอักษรได้เพียงตัวเดียว แต่ item ใน List สามารถเป็นอะไรก็ได้ เช่นอาจจะเป็น ตัวอักษร หรือ string ข้อความ หรือ ตัวเลข เช่น Integer หรือ Float ก็ได้ และอาจจะมี List เป็น item ของ List ได้ด้วยเช่นกัน ดังตัวอย่างในรูปด้านล่าง

```
>>> alist = ['Hello', 1, 3.14, [100, 'a']]
```

String จะถูกล้อมรอบด้วย single quote, double quote หรือ triple quote ก็ได้ แต่ List จะล้อมรอบด้วยเครื่องหมายเปิดปิด Square Brackets [ ]

```
>>> astring = 'Hello World!'
>>> alist = list('Hello World!')
>>> alist
['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '!']
>>> alist[0]
'H'
>>> astring[0]
'H'
>>> alist[-1]
'!'
>>> astring[-1]
'!'
```

จากรูปด้านบน จะเห็นได้ว่า เราสามารถใช้ List เก็บข้อความตัวอักษรลักษณะเดียวกันกับ String ได้ และสามารถใช้ index เพื่อเข้าถึง item ที่อยู่ใน List ได้เช่นเดียวกับ String

แต่ละ item ใน List จะถูกแยกออกจากกันด้วยเครื่องหมาย comma แต่ item ใน String ซึ่งก็คือตัวอักษรแต่ละตัวจะถูกเขียนติดกันไป

list เป็นชื่อ function ในการแปลงข้อมูลที่เป็นลำดับเช่น string ให้เป็น list ได้ ดังนั้น จึงไม่ควรตั้งชื่อตัวแปรว่า **list** อีก

```
>>> list1 = ['Hello', 1, 3.14, [2, 3], []]
>>> len(list1)
5
>>> len([])
0
```

ในรูปด้านบน list1 จะเป็น List ที่มี item อยู่ 5 items โดย item แรกคือ 'Hello' item ที่ 2 คือ 1 item ที่ 3 คือ 3.14 item ที่ 4 คือ list ที่มีตัวเลข 2 และ 3 และ item ที่ 5 เป็น list ที่ไม่มี item อยู่ภายในเลย เราจะเรียก List ที่ไม่มี item อยู่เลยว่า Empty List หรือ List ว่างนั่นเอง โดย List ว่างจะมีขนาดของ List เป็น 0

List is Mutable , Str is immutable

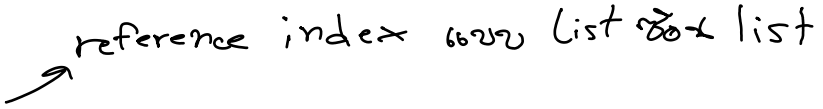
สมมุติว่าเราจะจัดเก็บวิธีอ่านตัวเลข 1-5 ของภาษาญี่ปุ่นลงใน list

Number	Sino-Japanese reading
1	いち (ichi)
2	に (ni)
3	さん (san)
4	し、よん (shi, yon)
5	ご (go)
6	ろく (roku)
7	しち、なな (shichi, nana)
8	はち (hachi)
9	く、きゅう (ku, kyuu)
10	じゅう (juu)
0	れい、ゼロ、マル (rei, zero, maru)

(รูปจาก https://www.mondly.com/blog/2019/11/22/count-in-japanese-a-complete-guide-to-japanese-numbers/)

ตัวอย่างเช่น เลข 1 จะอ่านว่า ichi เลข 2 จะอ่านว่า ni เป็นต้น

```
>>> jp_numlist = ['ichi', 'ni', 'san', 'yon', 'go']
>>> jp_numlist[3] = 'shi'
>>> jp_numlist
['ichi', 'ni', 'san', 'shi', 'go']
>>> jp_numlist[3] = ['shi', 'yon']
>>> jp_numlist
['ichi', 'ni', 'san', ['shi', 'yon'], 'go']
>>> jp_numlist[0]
'ichi'
>>> jp_numlist[3]
['shi', 'yon']
>>> jp_numlist[3][0]
'shi'
>>> jp_numlist[3][1]
'yon'
>>> jp_numlist[4]
'go'
>>> jp_numlist[5]
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    jp_numlist[5]
IndexError: list index out of range
```



ในรูปด้านบน เริ่มต้น jp_numlist จะเก็บคำอ่านตัวเลข 1-5 ในภาษาญี่ปุ่น จะสังเกตได้ว่า เลข 4 ในภาษาญี่ปุ่นสามารถอ่านออกเสียงได้ 2 แบบ คือ 'Shi' และ 'Yon' ในตอนแรก เลข 4 ได้เก็บคำอ่านว่า 'Yon' ไว้

เนื่องจากว่า List เป็น Mutable Type ดังนั้นเราสามารถเปลี่ยนค่าใน list jp_numlist ของเลข 4 จากเดิม 'Yon' เป็น 'Shi' ได้ ด้วย jp_numlist[3] = 'Shi' จะสังเกตเห็นว่า index จะมีค่าเป็นเลข 3 แต่เรากำลังหมายถึง item ลำดับที่ 4 เนื่องจาก index จะมีค่าเริ่มจาก 0 เสมอ

ข้อควรระวังอีกข้อคือ String เป็น Immutable Type ซึ่งจะแตกต่างกับ List ดังนั้นเราไม่สามารถเปลี่ยนแปลงค่าใน String ได้ ดังที่ได้กล่าวไปในบทที่แล้ว

เนื่องจากว่าเราสามารถจัดเก็บข้อมูลเป็น List เป็น item ใน List ได้ ดังนั้น เราจึงควรเก็บทั้งการอ่านแบบ 'Shi' และ 'Yon' ไว้ใน List jp_numlist ด้วย ดังในรูปตัวอย่างด้านบน และหากเราต้องการถึงข้อมูล 'Shi' เราสามารถเข้าถึงได้ด้วย index 2 ชั้น เป็น '[3][0]' และ 'Yon' เข้าถึงได้ด้วย index '[3][1]' ตามลำดับ

หากเราพยายามอ้างถึง item ที่ไม่มีอยู่ด้วยหมายเลข index ที่เกินกว่าช่วงที่เป็นไปได้ของ List นั้น เราจะได้รับ Error ประเภท IndexError

List Operators

```
>>> 'Hello' + 'World'
'HelloWorld'
>>> ['ichi', 'ni'] + ['san']
['ichi', 'ni', 'san']
>>> 'Hello'*3
'HelloHelloHello'
>>> ['ichi', 'ni']*3
['ichi', 'ni', 'ichi', 'ni', 'ichi', 'ni']
>>> 'H' in 'Hello'
True
>>> 'h' in 'Hello'
False
>>> 'ichi' in ['ichi', 'ni']
True
>>> 'i' in ['ichi', 'ni']
False
```

เราสามารถใช้ operator เช่น '+' เพื่อการ concatenate และ '*' และ 'in' ในลักษณะเดียวกับที่ใช้ใน String กับ List ได้เช่นเดียวกัน ดังตัวอย่างในรูปด้านบน นอกเหนือจาก operator ข้างต้น 3 แบบ แล้ว ยังมี operator อื่นๆที่สามารถใช้กับ list ได้อีก เช่น del ซึ่งจะได้กล่าวถึงต่อไป

Traversing a List

การเข้าไปใน List เพื่อเข้าถึงข้อมูล item แต่ละตัวใน List ตั้งแต่ item แรก จนถึง item สุดท้ายใน List เรียกว่า **Traversing**

การเข้าถึงโดยทั่วไปมี 2 แบบ

แบบที่ 1 คือการเข้าถึงเพื่อการอ่านข้อมูลของแต่ละ item ใน List นั้น โดยไม่มีการเปลี่ยนแปลงแก้ไข item นั้น

แบบที่ 2 คือการเข้าถึงเพื่อต้องการแก้ไขข้อมูลใน item ใน List นั้น

โดยปกติแล้ว เรามักจะใช้ **For loop** ในการ Traversing เข้าไปใน List ถ้าเป็นการเข้าถึงแบบที่ 1 จะใช้การเขียนด้วย **For loop** แรกในรูปด้านล่างนี้

```
>>> numlist = [1, 2, 3, 4, 5]
>>> for num in numlist:
        print(num)


1
2
3
4
5
>>> for index in range(len(numlist)):
        numlist[index] = numlist[index]**2


>>> numlist
[1, 4, 9, 16, 25]
```

แต่หากเป็นการเข้าไปใน item ของ List เพื่อแก้ไขข้อมูล จะใช้ index เป็นหมายเลขอ้างอิง ดังใน **For loop** ที่สองในรูปด้านบนนี้ numlist จะมีการแก้ไขค่าของ item โดยทุกตัวจะมีค่าใหม่ที่ถูกเปลี่ยนแปลงไปเป็นค่าเลขยกกำลังของค่าเดิม

List Slice

list slice มีการทำเหมือนกับ string slice แต่เนื่องจาก list เป็น Mutable type แต่ string เป็น Immutable Type ดังนั้น List จึงสามารถทำการแก้ไขค่าใน list โดยใช้ List slice ได้ ดังตัวอย่างด้านล่าง

```
>>> t = [1, 2, 3, 4, 5]
>>> t[1:3] = [20, 30]
>>> t
[1, 20, 30, 4, 5]
```

List Methods

list มี method ที่สามารถกระทำกับ list นั้น เช่นเดียวกับ method ที่มีในข้อมูลชนิดอื่นๆเช่น String

ในที่นี้จะแนะนำ methods ที่สำคัญ เริ่มจาก method สำหรับการนำข้อมูลไปเก็บใน list ซึ่งสามารถทำได้หลายวิธี (method)

วิธีที่ 1 หากมี item (object) ที่ต้องการเพิ่มเข้าไปใน list ที่ ให้ใช้ method append ดังตัวอย่างในรูปด้านล่าง

```
>>> list1 = ['a', 'b', 'c']
>>> list1.append('d')
>>> list1
['a', 'b', 'c', 'd']
```

เนื่องจาก list เป็น Mutable Type ดังนั้น การเขียนลักษณะด้านล่างจึงเป็นสิ่งที่ผิด เนื่องจากค่า list1 จะมีค่าเป็น None เนื่องจาก list1.append('d') ไม่ได้มีการ return ค่าใดๆกลับไป

```
>>> list1 = ['a', 'b', 'c']
>>> list1 = list1.append('d')
>>> list1
>>> type(list1)
<class 'NoneType'>
```

วิธีที่ 2 หากมี list2 หรือ iterable อื่นๆ เช่น tuple ที่ต้องการเพิ่มเข้าไปที่ด้านหลังของ list1 ให้ใช้ method extend ดังตัวอย่างในรูปด้านล่าง

```
>>> list1
['a', 'b', 'c', 'd']
>>> list2 = ['e', 'f']
>>> list1.extend(list2)
>>> list1
['a', 'b', 'c', 'd', 'e', 'f']
>>> list2
['e', 'f']
```

Method ที่กระทำกับ object หรือในที่นี้คือ list จะมีผลกับ object นั้น หรือ list นั้น โดยตรง เนื่องจาก list เป็น Mutable Type ดังนั้นจะสังเกตเห็นได้ว่า ค่า list1 มีการเปลี่ยนแปลงไปจากเดิมในรูปด้านบน

ความแตกต่างระหว่าง append กับ extend สามารถดูได้จาก Call Stack Visibility ของ append กับ extend ดังรูปด้านล่าง

```
>>> a = ['a', 'b', 'c', 'd']
>>> a.append(
            (object, /)
            Append object to the end of the list.
```

```
>>> a = ['a', 'b', 'c', 'd']
>>> a.extend(
            (iterable, /)
            Extend list by appending elements from the iterable.
```

จะเห็นได้ว่า argument ของ append คือ object แต่ argument ของ extend คือ iterable ตัวอย่างข้อมูลประเภท iterable ได้แก่ string, list, tuple, set และ dict เป็นต้น โดย extend จะทำการ iterate เข้าใน item แต่ละ item ใน iterable นั้น และทำการเสมือนว่า append ค่า item แต่ละ item ใน list ดังแสดงในรูปด้านล่าง

```
>>> a = ['a', 'b', 'c', 'd']
>>> a.extend("e")
>>> a
    ['a', 'b', 'c', 'd', 'e']
>>> a.extend("fgh")
>>> a
    ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

หาก Iterable เป็น dictionary เฉพาะ key ของ dictionary เท่านั้น ที่จะถูก extend เข้าไป

วิธีที่ 3 เป็นการ insert ค่าเข้าไปใน list ตามตำแหน่งที่ต้องการ วิธีนี้จะต่างจาก append และ extend ที่จะเพิ่มค่าเข้าไปด้านท้ายของ list เท่านั้น

```
>>> list1 = ['a', 'b', 'c', 'd']
>>> list1.insert(2, 'e')
>>> list1
['a', 'b', 'e', 'c', 'd']
>>> list1.insert(0, 'e')
>>> list1
['e', 'a', 'b', 'e', 'c', 'd']
```

วิธีที่ 4 ไม่ใช่การใช้ list method แต่เป็นการใช้ operator '+' เพื่อ concatenate list ซึ่งจะมีผลลัพธ์คล้ายๆกับการใช้ method extend แต่ให้ระวังว่า การใช้ '+' เราจะต้องมีตัวแปรอีกตัวมารับค่าผลลัพธ์ของการ '+' ในที่นี้คือ list3 ในรูปด้านล่าง และ list1 กับ list2 จะไม่มีการเปลี่ยนแปลงค่าใดๆ

```
>>> list1 = ['a', 'b', 'c']
>>> list2 = ['e', 'f']
>>> list3 = list1 + list2
>>> list3
['a', 'b', 'c', 'e', 'f']
>>> list1
['a', 'b', 'c']
>>> list2
['e', 'f']
```

หากต้องการลบ item ที่มีอยู่ใน list สามารถทำได้หลายวิธี

วิธีที่ 1 ใช้ pop method วิธีนี้จะต้องทำรู้ตำแหน่ง index ที่ต้องการจะลบ item ในที่นี้สมมุติว่าเรารู้ว่า 'b' คือ item ที่เราต้องการจะลบออกนั้น อยู่ในตำแหน่ง index ที่ 1 ดังตัวอย่างในรูปด้านล่าง

```
>>> list1 = ['a', 'b', 'c']
>>> x = list1.pop(1)
>>> list1
['a', 'c']
>>> x
'b'
```

วิธีที่ 2 ใช้ method remove วิธีนี้ไม่จำเป็นต้องรู้ว่า index ของ item ที่ต้องการจะลบ แต่รู้ว่าสิ่งที่ต้องการจะลบคือ 'b' แต่หากว่ามี 'b' มากกว่า 1 ที่ใน list การใช้ method remove จะลบแค่ 'b' ตัวแรกที่เจอเท่านั้น ดังตัวอย่างในรูปด้านล่าง

```
>>> list1 = ['a', 'b', 'c', 'd']
>>> list1.remove('b')
>>> list1
['a', 'c', 'd']
```

วิธีที่ 3 ไม่ใช่ list method แต่เป็นการใช้ del operator ข้อดีของวิธีนี้คือ เราสามารถลบ item ใน list ได้มากกว่า 1 ตัว โดยการกำหนด list slice เวลาใช้ del operator ดังรูปด้านล่าง

```
>>> list1 = ['a', 'b', 'c']
>>> del list1[1:3]
>>> list1
['a']
```

หากต้องการลบทุก `item` ที่อยู่ใน `list` ออกไปทั้งหมด สามารถใช้ method `clear` เช่น `list1.clear()` หรือใช้ `del list1[:]`

นอกจาก method append, extend, pop และ remove ที่กล่าวไปแล้ว ยังมี method อื่น ที่มีการใช้งานบ่อยๆ ได้แก่ index, count และ reverse

Lists and Strings

เราสามารถเปลี่ยนจาก String ไปเป็น List ได้ด้วย function list และสามารถเปลี่ยนจาก list ไปเป็น string ได้ด้วย method join ซึ่งเป็น method ของ String

```
>>> astring = 'Hello'
>>> alist = list(astring)
>>> alist
['H', 'e', 'l', 'l', 'o']
>>> bstring = ''.join(alist)
>>> bstring
'Hello'
```

ในที่นี้ empty string '' จะถูกใช้ในการเชื่อม item ใน alist เข้าด้วยกันเพื่อสร้างเป็น bstring ดังแสดงในรูปด้านบน

```
>>> bstring = '-'.join(alist)
>>> bstring
'H-e-l-l-o'
```

หากเปลี่ยน empty string เป็น '-' ดังตัวอย่างด้านบน item ใน alist ค่า จะเชื่อมกันโดยมี '-' คั่นระหว่าง item ทั้งหมด เพื่อสร้าง string ใหม่ให้กับ bstring ดังรูปด้านบน

function list จะทำการแยก string ออกเป็นตัวอักษรแต่ละตัว แต่หากต้องการแยกเป็นคำ หรือแยกตามตำแหน่งที่ต้องการ สามารถทำได้โดยใช้ method split ดังตัวอย่างด้านล่าง

```
>>> astring = 'Hello World today'
>>> list1 = astring.split()
>>> list1
['Hello', 'World', 'today']
>>>
>>> email = 'test@google.com'
>>> list1 = email.split('@')
>>> list1
['test', 'google.com']
```

Sorting

การเรียงลำดับข้อมูลใน LIst เป็นสิ่งที่เกิดขึ้นบ่อยมากในการทำโจทย์ มีหลายวิธีในการเรียงลำดับข้อมูลใน List

วิธีที่ 1 ใช้ method sort ดังแสดงในรูปด้านล่าง การเรียงลำดับจะเป็นจากน้อยไปมาก เป็นค่าเริ่มต้น หากต้องการเรียงจากมากไปน้อยให้ใส่ parameter reverse=True

```
>>> t = [3, 4, 1, 2, 5]
>>> t.sort()
>>> t
[1, 2, 3, 4, 5]
>>> t = [3, 4, 1, 2, 5]
>>> t.sort(reverse=True)
>>> t
[5, 4, 3, 2, 1]
```

การใช้ method sort จะเป็นการเปลี่ยนค่าของ list ไป ดังนั้นหากเราต้องการเก็บค่าของ list ให้ทำการ copy list นั้นเก็บไว้ก่อน (ด้วยการทำ list slice) ดังตัวอย่างในรูปด้านล่าง

```
>>> t = [3, 4, 1, 2, 5]
>>> t2 = t[:]
>>> t.sort()
>>> t
[1, 2, 3, 4, 5]
>>> t2
[3, 4, 1, 2, 5]
```

วิธีการ copy list อีกวิธีหนึ่งคือการใช้ copy method โดยสามารถเขียน `t2 = t.copy()` แทนการใช้ list slice ในรูปด้านบนได้

เนื่องจาก List เป็น Mutable Type ดังนั้น method ของ List จะเป็นการเปลี่ยนค่าของ List เอง (ไม่ได้มีการสร้าง List ใหม่) ดังนั้นหากเราต้องการเรียงข้อมูลโดยการเขียนแบบด้านล่างจะให้ผลที่ผิด เนื่องจาก t จะมีค่าเป็น None

```
>>> t = [3, 4, 1, 2, 5]
>>> t = t.sort()
>>> t
>>> type(t)
<class 'NoneType'>
```

วิธีที่ 2 ใช้ sorted function ตัวอย่างการใช้งาน sorted function จะเป็นไปตามรูปด้านล่างนี้

```
>>> t = [3, 4, 1, 2, 5]
>>> t2 = sorted(t)
>>> t2
[1, 2, 3, 4, 5]
>>> t
[3, 4, 1, 2, 5]
>>> t2 = sorted(t, reverse=True)
>>> t2
[5, 4, 3, 2, 1]
>>> t
[3, 4, 1, 2, 5]
```

sorted function เป็น Fruitful function ดังนั้นเราจึงต้องมีการสร้างตัวแปรมารับค่า list ที่ได้มีเรียงลำดับ ที่ได้สร้างขึ้นมาใหม่ ดังในรูปตัวอย่างด้านบน

ข้อดีของการใช้ sorted function อีกอย่างคือ เราสามารถใช้ sorted function ในการ sort ข้อมูลลำดับนอกเหนือจาก list เช่น tuple และ set ได้ด้วย (ข้อมูลชนิด set จะได้กล่าวถึงในบทถัดไป)

Objects and Values

ในการเขียนโปรแกรม สิ่งที่เราสร้างขึ้นมาจะเรียกว่า object หรือ วัตถุ โดยที่วัตถุนั้น อาจจะมี ค่า หรือ value อยู่

ยกตัวอย่างในวัตถุในโลกความเป็นจริง เช่น หนังสือ เป็นวัตถุ เราอาจจะมีหนังสือ 2 เล่ม เล่มที่ 1 ก็เป็น object นึง เล่มที่ 2 ก็เป็นอีก object นึง กล่าวคือเป็นคนละ object กัน แต่อย่างไรก็ตาม หนังสือทั้ง 2 เล่ม นี้อาจจะมีค่าเดียวกันได้ (same value) เช่น เป็นหนังสือชื่อ PSIT Book เหมือนกัน กล่าวคือมีเนื้อหาเหมือนกันทุกประการ

```
>>> a = 'banana'
>>> b = 'banana'
>>> b == a
True
>>> b is a
True
```

ดังรูปด้านบน เราเห็นว่ามี 'banana' อยู่ ซึ่ง a และ b ชี้ไปถึง คำถามคือ 'banana' ที่ a กับ b ชี้ไปนี้เป็น object เดียวกันหรือไม่ เมื่อเราใช้ operator is เพื่อตรวจสอบดูจะพบว่า a และ b เป็น object เดียวกัน (True)

กล่าวคือ String object ที่ a และ b ชี้ไปคือ object เดียวกัน และมีค่า (value) เป็น 'banana'

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a == b
True
>>> a is b
False
```

แต่ในกรณีที่ object นั้นเป็น list ดังแสดงในรูปด้านบน จะเห็นได้ a และ b มีค่าเดียวกัน (a == b เป็น True) แต่ ไม่ใช่ object เดียวกัน เพราะ (a is b เป็น False) หรือกล่าวอีกแบบคือ [1, 2, 3] ที่ a ชี้ไป และ [1, 2, 3] ที่ b ชี้ไป คือคนละ object แต่มีค่าเหมือนกันคือ [1, 2, 3]

ดังนั้นหากเราแก้ไขค่าของ object ที่ a ชี้ไป จะไม่มีผลกับ object ที่ b ชี้ไป ดังตัวอย่างในรูปด้านล่าง

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
>>> b
[1, 2, 3]
```

แต่ถ้าหากเขียนตามแบบในรูปด้านล่างนี้ คือให้ b = a เป็นการกำหนดให้ b ชี้ไปที่ที่ a ชี้ที่ object นั้นอยู่ กล่าวคือ ทั้ง a และ b จะชี้ไปที่ object เดียวกัน สังเกตได้จาก b is a มีค่าเป็น True

กรณีที่ object ใดๆมีการอ้างถึง หรือชี้ไปโดยใช้ชื่อ (ตัวแปร) มากกว่า 1 ตัว ดังเช่นในกรณีนี้ เราจะกล่าวได้ว่า object นั้นได้ถูก aliased หรือมีนามแฝง

เมื่อ a และ b อ้างถึงหรือชี้ไปที่ object เดียวกัน ถ้า a แก้ไขข้อมูลหรือค่าของ object นั้น ก็จะทำให้การเปลี่ยนแปลงนั้นมีผลไปถึง b ที่อ้างถึงไปที่ object เดียวกันด้วย

```
>>> a = [1, 2, 3]
>>> b = a
>>> b
[1, 2, 3]
>>> b is a
True
>>> a.append(4)
>>> a
[1, 2, 3, 4]
>>> b
[1, 2, 3, 4]
```

การส่งค่า list ให้กับ function ผ่านทาง argument ไปยัง parameter ของ function นั้น เป็นการทำ alias ด้วย แสดงว่าการแก้ไขใดๆที่เกิดขึ้นกับ list ภายใน function นั้น จะส่งผลถึง list ของตัว caller ที่ส่งไปให้ function นั้นด้วย

ยกตัวอย่างเช่น

```
>>> def delete_head(t):
        del t[0]


>>> x = [1, 2, 3]
>>> delete_head(x)
>>> x
[2, 3]
```

ดังรูปด้านบน เมื่อ x ส่งค่าไปให้ function delete_head ผ่าน parameter t ทั้ง x และ t จะอ้างถึงหรือชี้ไปที่ object [1, 2, 3] เดียวกัน หรือที่เรียกว่า alias ดังนั้นเมื่อมีการลบ item แรกของ t ออกไป ก็จะส่งผลถึง x ด้วย จะเห็นได้ว่า x มีค่าเป็น [2, 3] หลังจากที่ส่งค่าไปให้ delete_head function

การทำ list slice แล้วมีการให้ค่า (assign) คือการสร้าง object ใหม่ขึ้นมา ยกตัวอย่างดังรูปด้านล่าง

```
>>> t = [1, 2, 3, 4, 5]
>>> x = t
>>> x is t
True
>>> t = t[1:]
>>> t
[2, 3, 4, 5]
>>> x
[1, 2, 3, 4, 5]
```

ดังนั้นถ้ามีการสร้าง object ใหม่ขึ้นมา โดยการทำ list slice ดังในตัวอย่างรูปด้านล่าง ตัวแปรจะทำการชี้ไปที่ object ใหม่นั้น ดังนั้นจะไม่ได้มีการแก้ไขเปลี่ยนแปลง object เดิมที่ t เคยชี้ไป เมื่อเรียก bad_delete_head(x) จะเห็นได้ว่า ค่า x ไม่มีการเปลี่ยนแปลงใดๆ

```
>>> def bad_delete_head(t):
        t = t[1:]


>>> x = [1, 2, 3]
>>> bad_delete_head(x)
>>> x
[1, 2, 3]
```

Tuples

Tuple มีการจัดเก็บข้อมูลเป็นลำดับเหมือนกับ String และ List สามารถเข้าถึงข้อมูลแต่ละตัวที่จัดเก็บซึ่งเรียกว่า item หรือ element เช่นเดียวกับ List โดยใช้หมายเลข index

ความแตกต่างที่สำคัญของ tuple กับ list คือ tuple เป็น Immutable Type และแทนที่จะใช้เครื่องหมาย Bracket [,] ล้อมรอบข้อมูลที่จัดเก็บแบบที่ใช้ใน List การล้อมรอบข้อมูลของ Tuple จะใช้เครื่องหมายวงเล็บ (,) และจะใช้เครื่องหมายวงเล็บล้อมรอบหรือไม่ก็ได้

```
>>> tup1 = (1, 2, 3, 'a', 'b')
>>> tup2 = 1, 2, 3, 'a', 'b'
>>> tup1 == tup2
True
>>> tup1[3]
'a'
>>> tup1[-1]
'b'
>>> tup1[0] = 0
Traceback (most recent call last):
  File "<pyshell#80>", line 1, in <module>
    tup1[0] = 0
TypeError: 'tuple' object does not support item assignment
```

เนื่องจาก tuple เป็น immutable type ดังนั้น เราจึงไม่สามารถเปลี่ยนแปลงค่า tup1[0] จาก 1 เป็น 0 ได้ ดังรูปด้านบน

อย่างไรก็ตามแม้ว่า Tuple จะเป็น immutable type เราสามารถเก็บข้อมูล Mutable เช่น List เป็น Element ใน tuple ได้ และสามารถเปลี่ยนแปลงค่าของ element ที่เป็น mutable type ได้ด้วย แต่เราไม่สามารถ item assignment ค่าใหม่ หรือ ลบ item deletion ได้ เนื่องจาก Tuple เป็น Immutable type ดังตัวอย่างด้านล่าง

```
>>> tup1 = ([1,2,3], [4,5])
>>> tup1[0] = 'a'
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    tup1[0] = 'a'
TypeError: 'tuple' object does not support item assignment
>>> tup1[0].append(4)
>>> tup1
([1, 2, 3, 4], [4, 5])
>>> tup1[0].clear()
>>> tup1
([], [4, 5])
>>> del tup1[0]
Traceback (most recent call last):
  File "<pyshell#37>", line 1, in <module>
    del tup1[0]
TypeError: 'tuple' object doesn't support item deletion
>>>
```

เนื่องจากว่าเราสามารถเก็บค่า Mutable Element ใน Tuple ได้ ดังนั้นตัวอย่างด้านล่างแสดงให้เห็นว่า `tup1 is tup2` เป็น False เพราะเราสามารถเปลี่ยนแปลงค่า Mutable element ได้ นอกจากนี้เรายังสามารถทำ alias เช่นเดียวกับ List ได้ด้วย

```
>>> tup1 = ([1,2,3], [4,5])
>>> tup2 = ([1,2,3], [4,5])
>>> tup1 is tup2
    False
>>> tup1[0].append(4)
>>> tup1
    ([1, 2, 3, 4], [4, 5])
>>> tup2
    ([1, 2, 3], [4, 5])
>>>
>>> tup3 = tup2
>>> tup2[0].append(4)
>>> tup2
    ([1, 2, 3, 4], [4, 5])
>>> tup3
    ([1, 2, 3, 4], [4, 5])
>>>
```

กรณีที่ tuple มีแค่ item เดียว จำต้องมี comma ตามหลังด้วย ดังในรูปด้านบน ตัวอย่างล่าง ไม่เช่นนั้น tup1 ในตัวอย่างบนจะเป็นชนิด String

```
>>> tup1 = ('a')
>>> type(tup1)
<class 'str'>
>>> tup1 = ('a',)
>>> type(tup1)
<class 'tuple'>
```

เราสามารถสร้าง tuple จาก string ได้ ด้วยการใช้ function tuple ดังตัวอย่างด้านล่างนี้

```
>>> greeting = 'Hello'
>>> tup = tuple(greeting)
>>> tup
('H', 'e', 'l', 'l', 'o')
```

คุณลักษณะอื่นๆของ Tuple ก็จะเหมือนกับ List เช่นสามารถทำ Tuple slice ได้เช่นกัน

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> t2 = t[2:5]
>>> t2
('c', 'd', 'e')
>>> t
('a', 'b', 'c', 'd', 'e')
```

การใช้งาน tuple ที่ใช้บ่อยมากคือการใช้ tuple assignment เช่นตัวอย่างในรูปด้านล่าง เราสามารถสลับค่า a กับ b ได้โดยการเขียน tuple assigment เพียงบรรทัดเดียว a, b = b, a

นอกจากนี้หากเราต้องการ assign ค่า 1, 2, 3 ให้กับ a, b และ c ตามลำดับ เราสามารถใช้ tuple assigment โดยการเขียนเพียงบรรทัดเดียว a, b, c = 1, 2, 3

```
>>> a = 5
>>> b = 10
>>> a, b = b, a
>>> a
10
>>> b
5
>>>
>>>
>>> a, b, c = 1, 2, 3
>>> a
1
>>> b
2
>>> c
3
```

เราสามารถใช้ tuple ในการคืนค่า return ค่าหลายๆค่าจาก function ได้ ดังตัวอย่างในรูปด้านล่าง เราสามารถให้ myfunction return ค่ากลับมา 2 ค่า คือ a+b และ a-b ให้กับตัวแปร x และ y ตามลำดับได้

```
>>> def myfunction(a, b):
        return a+b, a-b

>>> x, y = myfunction(10, 1)
>>> x
11
>>> y
9
```

เนื่องจาก Tuple เป็น Immutable type ดังนั้นจึงมี method เพียง 2 method ได้แก่ count และ index

zip and enumerate

```
>>> s = 'abc'
>>> t = [1,2,3]
>>> z = zip(s, t)
>>> z
<zip object at 0x1043e5580>
>>> list1 = list(z)
>>> list1
[('a', 1), ('b', 2), ('c', 3)]
>>> for item in list1:
...     print(item)
...
...
('a', 1)
('b', 2)
('c', 3)
>>> for a, b in list1:
...     print(a, b)
...
...
a 1
b 2
c 3
>>>
```

```
>>> s = 'abc'
>>> t = [1,2,3]
>>> z = zip(s, t)
>>> z
<zip object at 0x101bfdf80>
>>>
>>> for item in z:
...     print(item)
...
...
('a', 1)
('b', 2)
('c', 3)
>>>
>>> z
<zip object at 0x101bfdf80>
>>> for item in z:
...     print(item)
...
...
>>> print(list(z))
[]
>>>
```

```
>>> s = 'abc'
>>> for a, b in enumerate(s):
...     print(a, b)
...
...
0 a
1 b
2 c
>>> for a, b in enumerate(s, 1):
...     print(a, b)
...
...
1 a
2 b
3 c
>>>
```

Iterable vs. Iterator

```
>>> import collections.abc as c
>>> s = 'abc'
>>> t = [1,2,3]
>>> z = zip(s, t)
>>> k = 123
>>> isinstance(s, c.Iterable)
True
>>> isinstance(t, c.Iterable)
True
>>> isinstance(z, c.Iterable)
True
>>> isinstance(k, c.Iterable)
False
>>>
```

```
>>> next(s)
Traceback (most recent call last):
  File "<pyshell#428>", line 1, in <module>
    next(s)
TypeError: 'str' object is not an iterator
>>> next(t)
Traceback (most recent call last):
  File "<pyshell#429>", line 1, in <module>
    next(t)
TypeError: 'list' object is not an iterator
>>> next(z)
('a', 1)
>>> next(z)
('b', 2)
>>> next(z)
('c', 3)
>>> next(z)
Traceback (most recent call last):
  File "<pyshell#433>", line 1, in <module>
    next(z)
StopIteration
>>>
```

Note: Every iterator is also an iterable, but not every iterable is an iterator in Python

Sorting Again

การ sort นั้น สามารถ sort ข้อมูลที่เป็นชุดหรือข้อมูลลำดับได้เช่นกัน ยกตัวอย่างเช่นข้อมูลที่เป็น List ของ Tuple หากเราใช้ sorted function หรือ sort method การจัดเรียงจะเรียงจากข้อมูลแรกใน Tuple ก่อน หากข้อมูลแรกใน Tuple ที่เท่ากัน มีมากกว่า 1 ข้อมูล ก็จะเรียงข้อมูลที่ 2 ใน Tuple เป็นลำดับถัด ดังแสดงในรูปด้านล่าง

```
>>> t = [(7, 3), (2, 8), (3, 6), (7, 2)]
>>> t.sort()
>>> t
[(2, 8), (3, 6), (7, 2), (7, 3)]
>>>
>>> t = [(7, 3), (2, 8), (3, 6), (7, 2)]
>>> t = sorted(t)
>>> t
[(2, 8), (3, 6), (7, 2), (7, 3)]
```

จะเห็นได้ว่า `(7, 2)` และ `(7, 3)` มีข้อมูลในตัวแรกใน Tuple เท่ากันคือ 7 จึงเปรียบเทียบข้อมูลที่ 2 ใน Tuple ต่อ ซึ่งก็คือเปรียบเทียบค่า 2 กับค่า 3 เนื่องจากค่า 2 มีน้อยกว่า จึงเรียง `(7, 2)` มาก่อน `(7, 3)`

หากเราต้องการศึกษาการใช้งาน sort method เพิ่มเติม สามารถทำได้โดยให้แสดงข้อความตัวช่วยวิธีการใช้งาน sort method ของ List ดังรูปด้านล่าง

```
>>> help(list.sort)
Help on method_descriptor:

sort(self, /, *, key=None, reverse=False)
    Sort the list in ascending order and return None.

    The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
    order of two equal elements is maintained).

    If a key function is given, apply it once to each list item and sort them,
    ascending or descending, according to their function values.

    The reverse flag can be set to sort in descending order.
```

จะเห็นได้ว่า Method sort มี parameter ชื่อว่า `key` ซึ่งมีค่า default เป็น `None` และจากคำอธิบายเขียนว่า "If a key function is given, apply it once to each list item and sort them..., according to their fuction value" แสดงว่า `key` คือชื่อของ function ที่รับค่า item ใน list แต่ละตัวเข้าไป เพื่อเรียงตามค่าของผลของ function นั้น ที่เรียกว่า function value นั่นเอง ยกตัวอย่างเช่นในรูปด้านล่างนี้

```
>>> t = [-1, 2, 3, -5]
>>> t.sort()
>>> t
[-5, -1, 2, 3]
>>>
>>> t = [-1, 2, 3, -5]
>>> t.sort(key=abs)
>>> t
[-1, 2, 3, -5]
```

หากเรา sort ปกติ โดยไม่มีค่า key function ก็จะได้ผลตามที่คาดไว้ คือ -5 น้อยที่สุด ตามด้วย -1, 2, 3 ตามลำดับ แต่หากเรากำหนดให้ key function เป็น function abs ซึ่งก็คือ function absolute การ sort นั้นก็จะเรียงตามค่า absolute

กล่าวคือ -1 มีค่า absolute น้อยที่สุดคือ 1 เลยมาลำดับแรก ส่วน -5 มีค่า absolute เป็น 5 ซึ่งมีค่ามากที่สุดใน List นี้ จึงอยู่ลำดับสุดท้าย

ตัวอย่างด้านบน key function ที่ใช้เป็น abs() เป็น builts-in function จริงๆแล้วเราสามารถสร้าง function ขึ้นเอง และกำหนดให้เป็น key function ได้เช่นกัน ยกตัวอย่างในรูปด้านล่างนี้

```
>>> def myfunc(x):
        if x%2 == 1:
            return x
        else:
            return -x


>>> t = [-1, 2, 3, -5]
>>> t.sort(key=myfunc)
>>> t
[-5, 2, -1, 3]
```

myfunc() นี้กำหนดว่าหากเป็นเลขคี่ให้คืนค่าเลขคี่กลับไป แต่หากเป็นเลขคู่ให้คืนค่าลบของเลขคู่นั้นกลับไป ดังนั้น -5 เป็นเลขคี่จะมีค่า function value เป็น -5 ซึ่งน้อยที่สุด ส่วน 2 มีค่าเป็นเลขคู่ ก็จะคืนค่า function value เป็น -2 (เลขคู่ต้องคืนค่าติดลบ) ซึ่งน้อยเป็นลำดับรองลงมา ตามด้วย -1 ซึ่งเป็นเลขคี่ และ 3 ซึ่งเป็นเลขคี่ ซึ่งมีค่า function value เท่ากับตัวเลขนั้น

Json

ในการทำโจทย์บางข้อ input ที่รับเข้ามาจะมีลักษณะเป็น List แต่เมื่อรับด้วย input() จะกลายเป็น String ทั้งหมด ดังนั้นหากเราต้องการให้เป็น List จำเป็นจะต้อง import module json เข้ามาก่อน แล้วใช้ method loads เพื่อแปลงข้อมูลจาก string ให้กลายเป็น list ดังแสดงในรูปด้านล่าง

```
>>> x = input()
[1, 2, 3]
>>> x
'[1, 2, 3]'
>>> type(x)
<class 'str'>
>>> import json
>>> y = json.loads(x)
>>> y
[1, 2, 3]
>>> type(y)
<class 'list'>
```

item ที่อยู่ใน list ที่รับผ่าน input() อาจจะเป็น type อะไรก็ได้ เช่น string, int, float แต่ถ้า item เป็น string จะต้องใช้ double quote ห้ามใช้ single quote ดังตัวอย่างในรูปด้านล่าง สังเกตได้ว่า 'a', 'b' และ 'c' ใช้ single quote เมื่อแปลงเป็น list ด้วย json.loads จะเกิด error ขึ้น

```
>>> x = input()
['a', 'b', 'c']
>>> x
"['a', 'b', 'c']"
>>> y = json.loads(x)
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    y = json.loads(x)
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/json/__in
it__.py", line 357, in loads
    return _default_decoder.decode(s)
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/json/deco
der.py", line 337, in decode
    obj, end = self.raw_decode(s, idx=_w(s, 0).end())
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/json/deco
der.py", line 355, in raw_decode
    raise JSONDecodeError("Expecting value", s, err.value) from None
json.decoder.JSONDecodeError: Expecting value: line 1 column 2 (char 1)
```

ดังนั้น item ที่รับเข้ามาจาก input ใน list ควรใช้ double quote ดังแสดงในรูปด้านล่างนี้ สังเกตได้ว่า "a", "b" และ "c" ใน list ที่รับโดย input() ใช้ double quote

```
>>> x = input()
["a", "b", "c"]
>>> x
'["a", "b", "c"]'
>>> y = json.loads(x)
>>> y
['a', 'b', 'c']
>>> type(y)
<class 'list'>
>>> type(y[0])
<class 'str'>
```