

w06-Lec2

One-Dimensional Lists and Tuples - Part III

for 204111

by Kittipitch Kuptavanich

Map, Filter and Reduce

- ฟังก์ชัน `sum()` ดำเนินการบน **Element** ทุกตัวใน **List** แล้วให้ **Return Value** เป็นผลรวมของ แต่ละ **Element**
 - เราเรียกการดำเนินการโดยใช้ค่าของ **Element** หลาย ๆ ตัวใน **List** แล้วให้ผลลัพธ์เป็นค่าเพียงหนึ่งค่าว่า **Reduce**

```
>>> print(sum([2, 3, 5, 7, 11])) # 28
28
```

Map, Filter and Reduce [2]

- พิจารณาฟังก์ชัน `only_upper()` ที่สร้าง List ใหม่ จากคำใน List ที่เป็น Upper Case เท่านั้น

```
>>> only_upper(['A', 'bat', 'IS', 'Near'])  
['A', 'IS']
```

- Operation ในลักษณะนี้เรียกว่า **Filter** เนื่องจากเลือกเฉพาะสมาชิกบางตัวจาก List และคัดกรองบางตัวทิ้งไป

Map, Filter and Reduce [3]

ทุกตัว เริ่มด้วยตัวพิมพ์ใหญ่

- พิจารณาฟังก์ชัน `capitalize_all()` ที่สร้าง List ใหม่ที่ประกอบด้วยสมาชิกของเดิมทุกตัวในรูป Capitalized

```
>>> capitalize_all(['the', 'force', 'awakens'])  
['_The', '_Force', '_Awakens']
```

- Operation ในลักษณะนี้เรียกว่า **Map** เนื่องจากสมาชิกแต่ละตัวใน List ผลลัพธ์ เกิดจากการดำเนินการ (ในกรณีนี้ `str.capitalize()`) ลงบนสมาชิกแต่ละตัวของ List เดิม (1:1)

The `map()` Function [2]

- พิจารณาการหาความยาวหลักของสมาชิกใน List จำนวนเต็มบวก

```

>>> def digit_count(x):
    return len(str(x))

>>> a = [3197, 69, 73948, 8216, 4, 982660, 58]
>>> b = list(map(digit_count, a))
>>> b
[4, 2, 5, 4, 1, 6, 2]

>>> d = list(map(lambda x: len(str(x)), a))
>>> d
[4, 2, 5, 4, 1, 6, 2]

```

parameter

expression

ใช้ได้

ลองว่า สมาชิกใน a: เอา ค มาแทนที่ x

- `lambda` statement ใน Python มีหน้าที่เปลี่ยน **Parameter** และ **Expression** ให้เป็นฟังก์ชันที่ไม่มีชื่อ โดยฟังก์ชันจะมีหน้าที่คืนค่าที่ **evaluate** ได้ตาม Expression ที่ระบุ

The `map()` Function [3]

- เราสามารถใช้ `map()` กับ iterables อื่น ๆ ได้เช่นกัน (รวมถึง `set` และ `dict`)

```
# with strings
```

```
>>> list(map(lambda x: x.upper(), "happy"))
['H', 'A', 'P', 'P', 'Y']
```

```
# with ranges
```

```
>>> list(map(lambda x: x**2, range(1, 5)))
[1, 4, 9, 16]
```

```
# with tuples
```

```
>>> list(map(lambda x: 5*(x-32)/9, (122, 68, 23)))
[50.0, 20.0, -5.0]
```

The `map()` Function [4]

- ในกรณีการทำงานกับ 1 List ฟังก์ชัน `map()` จะทำงานกับฟังก์ชันที่มีหนึ่งพารามิเตอร์เท่านั้น

```
>>> a = [3197, 69, 73948, 8216, 4, 982660, 58]
>>> list(map(lambda x: len(str(x)), a))
[4, 2, 5, 4, 1, 6, 2]
```

1 parameter

- เราสามารถทำงานกับฟังก์ชันที่มี n พารามิเตอร์ได้
 - แต่จะต้อง `map` ลงบน List จำนวน n List เช่นกัน
 - กรณี List ยาวไม่เท่ากัน ผลลัพธ์จะยึดตาม List input ที่สั้นที่สุด

```
>>> # notice the difference in length
>>> a = [1, 2, 3]
>>> b = [4, 5]
>>> list(map(lambda x, y: x * y, a, b))
[4, 10]
```

Handwritten notes in Thai:

- Red arrows pointing from `a` and `b` to the `x` and `y` parameters in the lambda function.
- Red circles around `3` in `a` and `5` in `b`, with a red arrow pointing to the text "ไม่แสดง" (not displayed).
- Red text "แสดงเท่าที่มี" (display as much as there is) at the bottom.

The `map()` Function [5]

- พิจารณาการคำนวณค่าเบี่ยงเบนมาตรฐานของจำนวนจริง N จำนวนด้วยสูตร

$$s_N = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2}$$

```
>>> x = [1, 2, 3, 4]
>>> avg = sum(x)/len(x)
>>> avg
2.5
```

```
>>> (sum(map(lambda x: (x - avg)**2, x)) / len(x))**.5
1.118033988749895
```

The `map()` Function [6]

```
>>> # list of tuples of base and height
>>> b_h_list = [(1, 2), (3, 8), (20, 5)]

>>> def triangle_area(tuple_t):
...     b = tuple_t[0]
...     h = tuple_t[1]
...     return 0.5 * b * h
...
>>>
>>> list(map(triangle_area, b_h_list))
[1.0, 12.0, 50.0]
```

- ในกรณีที่เรามีความจำเป็นต้องใช้ตัวแปรมากกว่าหนึ่งตัวแปรในการคำนวณ เช่น กรณีต้องการหาพื้นที่ของสามเหลี่ยมดังตัวอย่าง เราสามารถกำหนด **parameter** ของฟังก์ชันที่จะใช้ในการ **map** เป็น **tuple** เพื่อทำงานกับ **list** ของ **tuple** ได้

The `map()` Function [7]

```
>>> # map is lazy
>>> def upper_len(x):
...     print(x.upper())
...     return len(x)
...
>>>
>>> map(upper_len, ['how', 'now', 'brown', 'cow'])
<map object at 0x7f848f3920e0>
# no print() output - lazy evaluation
>>>
>>> list(map(upper_len, ['how', 'now', 'brown', 'cow']))
HOW
NOW
BROWN
COW
[3, 3, 5, 3]
```

map() and range() – Example 1

```

>>> n = 5
>>> l_a = list(range(1, n + 1))
>>> print(l_a)
[1, 2, 3, 4, 5]
>>> l_b = list(map(lambda x: str(list(range(1, x+1))), l_a))
>>> print(l_b)
['[1]', '[1, 2]', '[1, 2, 3]', '[1, 2, 3, 4]', '[1, 2, 3, 4, 5]']
>>> l_b = list(map(lambda x: x.strip('[]'), l_b))
>>> result = '\n'.join(l_b).replace(',', ' ')
>>> print(result)
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

```

Diagram illustrating the execution of the code:

- The initial list `l_a` is `[1, 2, 3, 4, 5]`.
- The first `map` operation uses a `lambda` function `x: str(list(range(1, x+1)))` to transform each element of `l_a` into a string representation of a list of numbers from 1 to `x`.
- The resulting list `l_b` is `['[1]', '[1, 2]', '[1, 2, 3]', '[1, 2, 3, 4]', '[1, 2, 3, 4, 5]']`.
- The second `map` operation uses a `lambda` function `x: x.strip('[]')` to remove the square brackets from each string in `l_b`.
- The final `result` is a multi-line string where each line contains numbers from 1 to `x`, separated by spaces.

```

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

```

map() and range() – Example 2

```


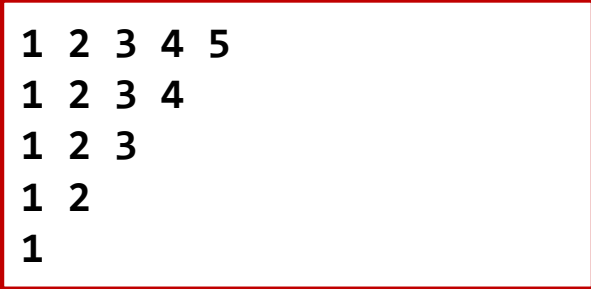
>>> n = 5
>>> l_a = list(range(n, 0, -1))
>>> print(l_a)
[5, 4, 3, 2, 1]

>>> l_b = list(map(lambda x: str(list(range(1, x+1))), l_a))
>>> print(l_b)
['[1, 2, 3, 4, 5]', '[1, 2, 3, 4]', '[1, 2, 3]', '[1, 2]', '[1]']

>>> l_b = list(map(lambda x: x.strip('[]'), l_b))

>>> result = '\n'.join(l_b).replace(',', ' ')
>>> print(result)
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1

```

map() and range() – Example 3

```

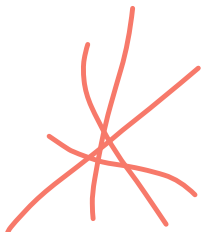


>>> n = 5

>>> a = list(range(1, n+1))
>>> print(a)
[1, 2, 3, 4, 5]

>>> b = list(map(lambda x: (('.'*(n-x)) + '*.*x').rstrip(), a))
>>> print(b)
['.', '.*', '.*.*', '.*.*.*', '.*.*.*.*']

>>> result = '\n'.join(b)
>>> print(result)
*
* *
* * *
* * * *
* * * * *

```

map() and range() – Example 4

```

>>> n = 5
>>> l_a = list(range(1, n+1))

>>> def helper(x):
...     start = (x**2 - x)//2 + 1
...     stop = start + x
...     return str(list(range(start, stop))).strip('[]')
...

>>> l_b = list(map(helper, l_a))
>>> print(l_b)
['1', '2, 3', '4, 5, 6', '7, 8, 9, 10', '11, 12, 13, 14, 15']
>>> result = '\n'.join(l_b).replace(',', ' ')
>>> print(result)
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15

```

```

1
2 3
4 5 6
7 8 9 10
11 12 13 14 15

```

The `filter()` Function

- Python มีฟังก์ชัน built-in `filter()` เช่นกัน

```
>>> def positive(x):    # To use with filter() the function
...     return x > 0    # must return True or False only

>>> a = [1, -2, -3, 4, 5]
>>> list(filter(positive, a))
[1, 4, 5]
```


Practice 1: Count Numbers w/ 8

- นับจำนวนของตัวเลขตั้งแต่ 300 ถึง 800 ที่หลักใดหลักหนึ่งมีเลข 8

```
>>> range_a = range(300, 800)
>>> f = filter(lambda x: '8' in str(x), range_a)
>>> len(f)
```

96

The `reduce()` Function

- ฟังก์ชัน `reduce()` อยู่ใน Module `functools`

```
>>> from functools import reduce
>>> a = [1, 4, 5, 3, 5]
>>> reduce(lambda x, y: x * y, a)
300
```

Accumulator
Default คือ Element ที่ 0

Element ที่ n
(ถ้าไม่ระบุ Accumulator
คือเริ่มดูจาก Element ที่ 1)

```
>>> a = [360, 20, 32]
>>> reduce(lambda x, y: math.gcd(x, y), a)
>>> 4
```

```
>>> b = ['leopard', 'godzilla', 'ape', 'flower', 'deer']
>>> reduce(lambda x, y: y if len(y) > len(x) else x, b)
'godzilla'
```

ระบุค่าเริ่มต้นของ x

```
>>> reduce(lambda x, y: max(x, len(y)), b, -1)
8
```

Practice 2: Comparing Lists

- เปรียบเทียบ List ที่มีสมาชิกเป็น **float**

```
>>> list_a = [1, 3, 5]
>>> list_b = [1.0001, 3.0001, 5.0001]
>>> from math import isclose
>>> e = 10**-3
>>> result = map(lambda x,y : isclose(x,y,abs_tol=e),
list_a, list_b), list_a, list_b)
>>> list(result)
[True, True, True]
>>> all(list(result))
True
```

ฟังก์ชัน **all()** จะ return **True**
เมื่อทุกสมาชิกเป็น **True**

The `all()` and `any()` Functions

```
05 list_a = "Joe", 10, 3.5
06
07 # Check if all variables are integers
08 if not all(map(lambda x: isinstance(x, int), list_a)):
09     print("Some variables are not integers")
10 else:
11     print("All variables are integer")
12
13
14 # Check if any variable is an integer
15 if not any(map(lambda x: isinstance(x, int), list_a)):
16     print("There are no integers")
17 else:
18     print("There are integers")
19
20
```

More on the `lambda` Function

```
>>> def my_abs(x):
...     if x < 0:
...         return -x
...     else:
...         return x

>>> a = [-4, 69, 73, -5, 4, 25]
>>> list(map(my_abs, a))
[4, 69, 73, 5, 4, 25]

>>> list(map(lambda x: -x if x < 0 else x, a)) # if-else
[4, 69, 73, 5, 4, 25]
```

- **Syntax**

`lambda` <args>: <return_val> `if` <cond> `else` <return_val>

More on the `lambda` Function [2]

```
>>> def classify(x):
...     if x in 'aeiou':
...         return 'V' # vowel
...     elif x == 'y':
...         return 'B' # both
...     else:
...         return 'C' # consonant

>>> letters = list('happy')
>>> list(map(lambda x: 'V' if x in 'aeiou' else ('B' if x
== 'y' else 'C'), letters)) # readability?
['C', 'V', 'C', 'C', 'B']
```

- Syntax

`lambda` *<args>*: *<return_val>* `if` *<cond>*

`else` (*<return_val>* `if` *<cond>* `else` *<return_val>*)

Misuse of `lambda` Functions

```
05 # The official python style guide PEP8, strongly
06 discourages the assignment of lambda expressions as shown
07 in the example below.
08
09 sum = lambda x, y: x + y
10 print(type(sum))          # <class 'function'>
11
12 x1 = sum(4, 7)
13 print(x1)                  # 11
14
15 # instead of this
16 func = lambda x, y, z: x*y + z
17
18 # it is recommended to write a one-liner function as
19 def func(x, y, z): return x*y + z
20
```

Map, Filter and Reduce [4]

	Map	Filter	Reduce
Module	<code>builtins</code>	<code>builtins</code>	<code>functools</code>
Syntax	<code>map(func, iter)</code>	<code>filter(func, iter)</code>	<code>reduce(func, iter)</code>
input/output ratio	$N \rightarrow N$	$N \rightarrow M$ ($M \leq N$)	$N \rightarrow 1$
Operation Type	individual element operation	individual element operation	pair-wise operation

zip and unzip

```
>>> s_id = ['701', '702', '703']
>>> score = [4.3, 5.2, 3.6]
>>> zipped = zip(s_id, score)
>>> type(zipped)
<class 'zip'>

>>> id_score = list(zipped)
>>> id_score
[('701', 4.3), ('702', 5.2), ('703', 3.6)]

>>> id = ['701', '702', '703']
>>> score1 = [4.3, 5.2, 3.6]
>>> score2 = [4.6, 5.7, 4.1]
>>> id_score = list(zip(id, score1, score2))
>>> id_score
[('701', 4.3, 4.6), ('702', 5.2, 5.7), ('703', 3.6, 4.1)]
```

zip and unzip [2]

```
>>> # unzipping
>>> a, b, c = zip(*id_score)
>>> a
('701', '702', '703')
>>> b
(4.3, 5.2, 3.6)
>>> c
(4.6, 5.7, 4.1)
>>> id_score
[('701', 4.3, 4.6), ('702', 5.2, 5.7), ('703', 3.6, 4.1)]
```

ใช้เครื่องหมาย * เพื่อระบุงการ unzip

```
>>> x = [['a', 1], ['b', 2]] # with lists
>>> x1, x2 = zip(*x)
>>> x1
('a', 'b')
>>> x2
(1, 2)
```

zip and unzip [3]

```
>>> # using zip and map
>>> a = ['Peter', 'Tony', 'Bruce', 'Clark']
>>> b = ['Spiderman', 'Ironman', 'Batman', 'Superman']

>>> c = zip(a, b)
>>> list(c)
[('Peter', 'Spiderman'), ('Tony', 'Ironman'), ('Bruce',
'Batman'), ('Clark', 'Superman')]

>>> c
<zip object at 0x7fb703703340>

>>> list(map(lambda x: _____, c))
['Peter - Spiderman', 'Tony - Ironman', 'Bruce - Batman',
'Clark - Superman']
```

Sorting Basics (Recap)

- แบบ Non-destructive

```
>>> sorted([5, 2, 3, 1, 4])  
[1, 2, 3, 4, 5]
```

- แบบ Destructive

```
>>> a = [5, 2, 3, 1, 4]  
>>> a.sort()  
>>> a  
[1, 2, 3, 4, 5]
```

- Method `list.sort()` ใช้ได้เฉพาะกับ List เท่านั้น แต่ฟังก์ชัน `sorted()` ใช้ได้กับ Iterable ชนิดใดก็ได้

Key Functions

- เราสามารถระบุวิธีในการเรียงลำดับผ่านฟังก์ชัน ในรูปของพารามิเตอร์ **key** ได้
- พิจารณาการ **Sort**

```
>>> nums = [-16, -50, 47, -2, 33, -5, -12]
>>> sorted(nums)
[-50, -16, -12, -5, -2, 33, 47]
```

- Sort ด้วยค่า **Absolute**

```
>>> sorted(nums, key=abs)
[-2, -5, -12, -16, 33, 47, -50]
```

Key Functions [2]

- Sort ตามตัวอักษร

```
>>> sorted("This is a test string from Andrew".split())
['Andrew', 'This', 'a', 'from', 'is', 'string', 'test']
```

- Case-insensitive sort

```
>>> sorted("This is a test string from Andrew".split(),
key=str.lower)
['a', 'Andrew', 'from', 'is', 'string', 'test', 'This']
```

- การเรียงลำดับที่เกิดขึ้นจะเป็นเรียงตามค่าที่ได้จากฟังก์ชันที่ระบุชื่อผ่านพารามิเตอร์ **key** เช่นฟังก์ชัน **abs()** หรือ Method **str.lower()** (ไม่ต้องใส่วงเล็บ)

Key Functions [3]

```
>>> a = [3, -5, -2, 1, 45, -23]
```

```
>>> def square(x):  
    return x ** 2
```

```
>>> sorted(a, key=square)  
[1, -2, 3, -5, -23, 45]
```

```
>>> sorted(a, key=lambda x: x ** 2)  
[1, -2, 3, -5, -23, 45]
```

parameter

expression

Key Functions [4]

- พิจารณา List ของ Tuple ที่เก็บข้อมูล ชื่อ เกรด และอายุ ของนักเรียน

```
>>> student_tuples = [  
... ('john', 'A', 15),  
... ('jane', 'B', 12),  
... ('dave', 'B', 10),  
]
```


- หากต้องการ เรียงลำดับโดยตามอายุ (index ที่ 2) ในแต่ละ Tuple โดยใช้ **key** และ **lambda** จะต้องทำอย่างไร

```
sorted(student_tuples, key=_____)
```


Key Functions [5]


```
>>> student_tuples = [
...     ('john', 'A', 15),
...     ('jane', 'B', 12),
...     ('dave', 'B', 10),
... ]

# sort by age
>>> sorted(student_tuples, key=lambda student: student[2])
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```



- เนื่องจากมีความจำเป็นต้องใช้ฟังก์ชัน **key** ในลักษณะนี้บ่อยครั้ง Python มีฟังก์ชันใน **Operator Module** เพื่อทำหน้าที่นี้โดยเฉพาะ

```
>>> from operator import itemgetter
>>> sorted(student_tuples, key=itemgetter(2))
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```



Ascending and Descending

- เราสามารถระบุวิธีในการเรียงลำดับผ่านฟังก์ชัน จากน้อยไปมาก (Ascending) หรือมากไปน้อย (Descending) ได้ทั้งใน `list.sort()` และ ฟังก์ชัน `sorted()` ในรูปของพารามิเตอร์ `reverse`

```
>>> sorted([3, -5, -2, 1, 45, -23])
[-23, -5, -2, 1, 3, 45]

>>> sorted([3, -5, -2, 1, 45, -23], reverse=True)
[45, 3, 1, -2, -5, -23]

>>> id_score = [('701', 4.3, 4.6),
...             ('702', 5.2, 5.7),
...             ('703', 3.6, 4.1)]
>>> sorted(id_score, key=lambda x: x[1] + x[2], reverse=True)
[('702', 5.2, 5.7), ('701', 4.3, 4.6), ('703', 3.6, 4.1)]
```

Sorting with Multiple Criteria

```
>>> # a list of (name, id, score)
>>> student_tuples = [
...     ('john', '65305', 15.0),
...     ('jane', '64102', 10.5),
...     ('dave', '65222', 15.0)]

>>> # sort by score descending (max score first) then by id
>>> temp = sorted(student_tuples, key=lambda x: x[1])
>>> temp
[('jane', '64102', 10.0), ('dave', '65222', 15.0), ('john',
'65305', 15.0)]
>>> result = sorted(temp, key=lambda x: x[2], reverse=True)
>>> result
[('dave', '65222', 15.0), ('john', '65305', 15.0), ('jane',
'64102', 10.0)]
```

- ถ้ามีหลาย **criteria** แล้วต้อง **sort** มากกว่า 1 ครั้ง ให้ **sort** ด้วย **key** ที่มีความสำคัญน้อยก่อน (กรณีนี้สำคัญที่สุดคือคะแนน)

Sorting with Multiple Criteria [2]

```
>>> date_list = [(10, 12, 1996), (10, 11, 2004), (2, 12, 2022)]

>>> temp = sorted(date_list, key=lambda x: x[0])    # sort by day
>>> temp
[(2, 12, 2022), (10, 12, 1996), (10, 11, 2004)]

>>> temp = sorted(temp, key=lambda x: x[1])        # sort by month
>>> temp
[(10, 11, 2004), (2, 12, 2022), (10, 12, 1996)]

>>> result = sorted(temp, key=lambda x: x[2])      # sort by year
>>> result
[(10, 12, 1996), (10, 11, 2004), (2, 12, 2022)]
```

- เรียงลำดับการ sort จาก **key** ที่สำคัญน้อยไปมาก
 - ถ้าต้องการให้เรียงวันที่ตามลำดับก่อนหลังที่ถูกต้อง
 - ต้อง sort จาก **dd** -> **mm** -> **yyyy**

Reference

- <https://wiki.python.org/moin/HowTo/Sorting>
- <https://docs.python.org/3/howto/sorting.html>
- <https://docs.python.org/3/howto/functional.html?highlight=lambda>
- <https://www.geeksforgeeks.org/overuse-of-lambda-expressions-in-python/>
- <https://docs.python.org/3/library/stdtypes.html#tuple>
- <https://docs.python-guide.org/writing/gotchas>
- <https://www.geeksforgeeks.org/default-arguments-in-python/>
- <http://www.cs.cmu.edu/~15110/lectures/lec15-Arrays.pdf>
- <https://morioh.com/p/a9949a7305da?f=5c21fb01c16e2556b555ab32>
- Guttag, John V *Introduction to Computation and Programming Using Python, Revised*