

Chapter 5 Python Programming For Data Science part2

กิจกรรม 5: Data Structure

#Basic Data Structures ที่เราใช้กันบ่อย ใน Python

- **list:** [1,2,3,4,5]
- **tuple:** (1,2,3,4,5)
- **dictionary:** {'A': 1, 'B': 2, 'C': 3}

List

- ลิสต์ (list) คือรายการที่เราสามารถปรับเปลี่ยนรายการได้ตลอดเวลา
- ลิสต์ใช้สัญลัษณ์ [] คั่นด้วยเครื่องหมายคอมมา ,

เช่น ลิสต์ของความสูงนักเรียนในกลุ่ม 5 คน มีดังนี้

[163.5, 150.0, 167.0, 161.25, 170.0]

เราสามารถนำลิสต์มาเก็บในตัวแปรได้ ทำให้ตัวแปรเดียวสามารถเก็บค่าไว้ได้หลายค่า เช่น

```
# ลิสต์ความสูงของนักเรียนในกลุ่ม
```

```
student_heights = [163.5, 150.0, 167.0, 161.25, 170.0]
```

```
print('student heights =', student_heights)
```

```
↳ student heights = [163.5, 150.0, 167.0, 161.25, 170.0]
```

แบบฝึกหัด 5.1

ลองสร้างลิสต์จากรายการสมาชิกดังต่อไปนี้ [10, 20, 30, 40, 50] แล้วเก็บลงในตัวแปร my_list และสั่งพิมพ์ ตัวแปร my_list

เขียนโค้ด...

↳ [10, 20, 30, 40, 50]

ขนาดของลิสต์

เราสามารถหาขนาดของลิสต์ได้ด้วยคำสั่ง len ซึ่งย่อมาจากคำว่า length เช่น

```
len(student_heights)
```

แบบฝึกหัด 5.2

จงหาขนาดของลิสต์ในตัวแปร `my_list` ใส่ตัวแปร `my_length` แล้วพิมพ์ค่าของตัวแปรดังกล่าวออกหน้าจอ

```
my_list = [157, 158, 159, 160]
```

เขียนโค้ด...

↪ 4

การเข้าถึงสมาชิกภายในลิสต์

- เราสามารถเข้าถึงสมาชิกแต่ละตัวในลิสต์ด้วยการระบุตำแหน่งของสมาชิกที่เราต้องการ
- การนับตำแหน่ง (index) ของสมาชิก จะเริ่มต้นจาก 0 เสมอ
- เช่น `student_heights[0]` จะแทนสมาชิกตัวแรกของลิสต์ความสูงของนักเรียน
ในขณะที่ `student_heights[1]` จะแทนสมาชิกตัวที่ 2 ของลิสต์ ฯลฯ

เช่น

```
student_heights = [163.5, 150.0, 167.0, 161.25, 170.0]
```

```
print(student_heights[0])
```

```
print(student_heights[1])
```

```
print(student_heights[2])
```

```
↪ 163.5
```

```
150.0
```

```
167.0
```


หากเราระบุตำแหน่งเกินจากขนาดของลิสต์ คอมไพเลอร์จะฟ้องออกมาด้วยคำว่า `IndexError` ซึ่งแปลว่า ระบุค่าตำแหน่งเกินขนาดของลิสต์ เช่น

```
student_heights = [157, 159, 161, 162, 165]
```

```
print(student_heights[10])    # ทั้งๆ ที่ลิสต์ student_heights มีขนาดแค่ 5 ตัว
```

↳ `IndexError: list index out of range`

การเปลี่ยนแปลงสมาชิกภายในลิสต์

เราสามารถเปลี่ยนแปลงสมาชิกภายในลิสต์ โดยการใช้เครื่องหมาย = เหมือนกับการกำหนดค่าในตัวแปร เช่น เราสามารถเปลี่ยนสมาชิกตัวที่ 2 ของลิสต์ student_heights จากค่า 150.0 ให้เป็นค่า 180.0 ได้

```
student_heights = [163.5, 150.0, 167.0, 161.25, 170.0]
```

```
student_heights[1] = 180.0
```

```
print(student_heights)
```

```
↳ [163.5, 180.0, 167.0, 161.25, 170.0]
```

การเลือกบางส่วนของลิสต์ (slicing)

นอกจากเราจะสามารถเลือกสมาชิกบางตัวในลิสต์ได้แล้ว เรายังสามารถเลือกบางส่วนของลิสต์ได้ด้วย คำสั่งเลือกบางส่วน (slicing): ลิสต์[a : b] จะเลือกสมาชิกตั้งแต่ **index** ที่ a จนถึง **index** ที่ **b-1**

เช่น ถ้าเราต้องการเลือกสมาชิกตั้งแต่ตัวที่ 2 จนถึงตัวที่ 4 ของลิสต์ในตัวแปร student_heights ซึ่งก็คือ index ที่ 1 และ index ที่ 3

เราจะใช้คำสั่ง `student_heights[1:4]`

Python Index

0	1	2	3	4	5

index ใน Python เริ่มที่ค่าศูนย์ (0)

10	20	30	40	50	60
0	1	2	3	4	5

>> value

>> index

word[0] → 10
word[1] → 20
word[2] → 30

คำสั่งเลือกบางส่วน (slicing): `ลิสต์[a : b]` จะเลือกสมาชิกตั้งแต่ `index` ที่ `a` จนถึง `index` ที่ `b-1`

Slice [:]

10	20	30	40	50	60
0	1	2	3	4	5

`list[0:3]` → `[10, 20, 30]`
`list[0:4]` → `[10, 20, 30, 40]`
`list[3:5]` → `[40, 50]`

การเลือกบางส่วนของลิสต์ (slicing)

คำสั่งเลือกบางส่วนของลิสต์สามารถระบุจุดจบได้ด้วย ลิสต์ [a:]

เช่น เราจะเลือกสมาชิกตั้งแต่ตัวที่ 2 เป็นต้นไปในลิสต์ student_heights ด้วยคำสั่ง

```
student_heights[1:]
```

แบบฝึกหัด 5.3

เรามีลิสต์ `my_list` ที่มีสมาชิก 10 ตัวดังนี้

```
my_list = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

จงเลือกสมาชิกตัวที่ 3 จนถึงตัวที่ 7

เขียนโค้ด...

↳ [30, 40, 50, 60, 70]

จงเลือกสมาชิกตัวที่ 3 จนถึงตัวที่ 5

เขียนโค้ด...

↳ [30, 40, 50]

Tuple

tuple เหมือนกับ list เกือบทุกอย่าง ยกเว้น update ค่าใน tuple ไม่ได้ และใช้สัญกรณ์ วงเล็บ ()

สร้าง tuple

```
tuple_a = (1, 2, 3, 4, 5)
```

```
tuple_b = ('hi', 'toy', True, False, 200)
```

พยายามอัปเดตค่า tuple จะขึ้น error เพราะ update ค่าใน tuple ไม่ได้

```
tuple_a[0] = 100
```

↪ TypeError: 'tuple' object does not support item assignment

Dictionary

#สร้าง Dictionary (key-value pairs)

```
dict_a = {  
"John": 25,  
"Jack": 30,  
"Mary": 29  
}
```

#พิมพ์ค่า ออกจาก dict_a

```
print(dict_a["John"])
```

↳ 25

```
print(dict_a["Jack"] )
```

↳ 30

```
print(dict_a["Mary"] )
```

↳ 29

Update ค่าใน Dictionary

#สร้าง และ Update ค่าใน Dictionary

```
dict_a = {  
"John": 25,  
"Jack": 30,  
"Mary": 29  
}  
dict_a["Jack"] = 33
```

#เพิ่มค่าใหม่ใน Dictionary

```
dict_a["Toy"] = 28  
print(dict_a)  
↳ {'John': 25, 'Jack': 33, 'Mary': 29, 'Toy': 28}
```

Print Dictionary

#create dictionary

```
dict_a = {  
    "John": 25,  
    "Jack": 30,  
    "Mary": 29  
}
```

#print dictionary

```
print(dict_a.keys()) ↪ dict_keys(['John', 'Jack', 'Mary'])  
print(dict_a.values()) ↪ dict_values([25, 30, 29])  
print(dict_a) ↪ {'John': 25, 'Jack': 30, 'Mary': 29}  
print(dict_a["John"]) ↪ 25
```

Compare Main Data Structures

List

- Ordered
- Mutable

Tuple

- Ordered
- IMmutable

Dict

- UNordered
- Mutable

Ordered แปลว่า เรียงลำดับ สามารถ slice ด้วย index ได้
Mutable แปลว่า สามารถแก้ไขข้อมูลใน data structure นั้นได้

แบบฝึกหัด 5.4

เราเลือก เพื่อน มา 5 คน

1. ทำ List [] ของ ส่วนสูง เพื่อน 5 คน
2. ทำ Tuple () ของ น้ำหนัก เพื่อน 5 คน
3. print ส่วนสูง ของ เพื่อน คนที่ 2
4. print น้ำหนัก ของ เพื่อน คนที่ 3 ถึง 5

กิจกรรม 6: ข้อความ

ข้อความ (string) ก็เป็นข้อมูลรูปแบบหนึ่งที่เราสามารถประมวลผลได้และทดไว้ในตัวแปรได้เช่นเดียวกัน
เช่น เราสามารถเก็บข้อความ 'Hello' ไว้ในตัวแปร message ได้

```
message = 'Hello'
```

```
print(message)
```

```
↳ Hello
```

การต่อข้อความ

เราสามารถนำข้อความ 2 อันมาต่อกันได้ด้วยเครื่องหมาย +

เช่น เราจะต่อข้อความในตัวแปร message1 และ message2 เข้าด้วยกันได้ดังนี้

```
message1 = 'Hello '
```

```
message2 = 'World!'
```

```
message = message1 + message2
```

```
print(message)
```

 Hello World!

การทำซ้ำข้อความ

นอกจากนี้เรายังสามารถทำซ้ำข้อความได้ด้วยเครื่องหมาย *

เช่น ถ้าเราต้องการทำซ้ำข้อความ hello 3 รอบ

`3 * 'hello'`

↳ `'hellohellohello'`

แบบฝึกหัด 6.1

จงสร้างตัวแปร 2 ตัว

message1 เก็บข้อความ 'Hello'

message2 เก็บข้อความ 'World'

หลังจากนั้นให้ทำซ้ำ message1 จำนวน 3 ครั้ง และ message2 จำนวน 5 ครั้ง นำผลลัพธ์ทั้งสองอันมาต่อกัน แล้วนำมาเก็บในตัวแปร message3 และสั่งพิมพ์ตัวแปร message3

เขียนโค้ด...

↳ HelloHelloHelloWorldWorldWorldWorldWorld

ข้อความในฐานะของลิสต์

ในภาษา Python ข้อความคือลิสต์ของตัวอักษร
เราสามารถเข้าถึงตัวอักษรสมาชิกแต่ละตัวในข้อความได้

```
message = 'wow'  
print(message[0])  
print(message[1])  
print(message[2])
```



w

o

w

ข้อความในฐานะของลิสต์

นอกจากนี้ เรายังสามารถเลือกบางส่วนของข้อความมาได้ด้วย เหมือนอย่างที่เราเลือกบางส่วนของลิสต์ เช่น เราสามารถเลือกตั้งแต่ตัวอักษร index ที่ 1 จนถึง index ที่ 3 ได้ด้วยคำสั่งนี้

```
message = 'flowers'
```

```
message[1:4]
```

↳ 'low'

แต่ทั้งนี้เราไม่สามารถแก้ไขสมาชิกในข้อความได้

```
message[1] = 'E'
```

↳ TypeError: 'str' object does not support item assignment

แบบฝึกหัด 6.2

1.จงทำซ้ำข้อความ 'aloha' จำนวน 3 ครั้งแล้วเก็บไว้ในตัวแปร my_string และสั่งพิมพ์ตัวแปร my_string

เขียนโค้ด...

↳ alohaalohaaloha

2.จงเลือกบางส่วนของข้อความ girl='you are cute'

เริ่มจากสมาชิกตัวที่ 1 จนถึงตัวที่ 3 แล้วเก็บลงในตัวแปร my_love และสั่งพิมพ์ตัวแปร my_love

เขียนโค้ด...

↳ you

กิจกรรม 7: การวนซ้ำ

ในกิจกรรม 5 และ 6 ต้องใช้วิธี copy-paste แล้วแก้หมายเลขตำแหน่งของสมาชิกทีละตำแหน่ง ถ้าเกิดลิสต์มีสมาชิกจำนวนมากขึ้นมา เราก็คงจะ copy-paste กันเยอะแน่นอน เราสามารถใช้วิธีการวนลูป (loop) เพื่อทำกระบวนการเดิมซ้ำ ๆ ได้

ลูปฟอรั (for-loop)

ลูปฟอรั (for-loop) ใช้สำหรับทำกระบวนการเดิมซ้ำ ๆ กับสมาชิกแต่ละตัวในลิสต์ ตามลำดับ
รูปแบบคำสั่งของลูปฟอรัคือ

```
for <ตัวแปร> in <ลิสต์>:
```

```
    <กระบวนการ>
```

คำสั่งนี้จะแทนสมาชิกแต่ละตัวในลิสต์ด้วยตัวแปรที่กำหนด แล้วจึงทำซ้ำกระบวนการนี้จนกว่าจะใช้สมาชิกครบทุกตัวตามลำดับ

กฎการย่อหน้าของภาษาไพธอน

- 1) กระบวนการที่เราต้องการทำซ้ำในรูปทุกชนิด จะต้องย่อหน้าถัดเข้าไปทางขวาเสมอ
- 2) ขนาดของย่อหน้าที่นิยมกันคือ เคาะ spacebar 4 ครั้ง
- 3) เมื่อไรก็ตามที่ย่อหน้ากลับออกมาทางซ้าย จะถือว่าจบลูปแล้ว

ลูปฟอรั (for-loop)

สมาชิกแต่ละตัวของลิสต์ student_heights จะแทนด้วยตัวแปร height

```
student_heights = [163.5, 150.0, 167.0, 161.25, 170.0]
```

```
for height in student_heights:
```

```
    print(height) # พิมพ์ค่าของตัวแปร height ออกหน้าจอ
```

```
print('That is all')
```

↳ 163.5

150.0

167.0

161.25

170.0

That is all

แบบฝึกหัด 7.1

1. พิมพ์สมาชิกแต่ละตัวในลิสต์ `my_list` ออกหน้าจอโดยใช้ลูปฟอร์และคำสั่ง `print`

```
my_list = ['hello', 'my', 'name', 'is', 'Pug']
```

เขียนโค้ด...



hello

my

name

is

Pug

2.เพิ่มค่าสมาชิกแต่ละตัวในลิสต์ my_numbers ตัวละ 10 และพิมพ์ออกหน้าจอ

```
my_numbers = [50, 40, 30, 20, 10]
```

```
for number in my_numbers:
```

```
    number=number+10
```

```
    print(number)
```



60

50

40

30

20

ช่วงจำนวน (range)

บางครั้งเราจำเป็นต้องวนลูปฟอร์บนเลขตำแหน่ง index ด้วย

เราสามารถใช้คำสั่ง `range(a , b)` เพื่อสร้างช่วงจำนวนในลูปฟอร์ได้ โดย a จะเป็นค่า index เริ่มต้นและ b-1 จะเป็นค่าสุดท้าย เช่น

```
for i in range(1, 6):
```

```
    print('i =', i)
```

```
↳ i = 1
```

```
    i = 2
```

```
    i = 3
```

```
    i = 4
```

```
    i = 5
```

ช่วงจำนวน (range)

หากเราใส่ค่า a ไป เหลือเพียง range(b) ช่วงจำนวนจะเริ่มตั้งแต่ 0 และสิ้นสุดที่ b-1

```
for i in range(6):
```

```
    print('i =', i)
```



```
i = 0
```

```
i = 1
```

```
i = 2
```

```
i = 3
```

```
i = 4
```

```
i = 5
```

นอกจากนี้เรายังสามารถกำหนดขนาดของก้าวในการสร้างช่วงจำนวนได้
โดยใช้คำสั่ง `range(a , b, c)` เมื่อค่า `c` คือขนาดของก้าว

```
# ก้าวมีขนาดเท่ากับ 2
```

```
for i in range(0, 10, 2):
```

```
    print('i =', i)
```

```
↳ i = 0
```

```
    i = 2
```

```
    i = 4
```

```
    i = 6
```

```
    i = 8
```

ข้อดีของการใช้คำสั่ง range ที่กำหนดขนาดของก้าวได้ ก็คือ เราสามารถไล่ index ย้อนหลังได้ด้วย เช่น

```
# ก้าวมีขนาดติดลบ แปลว่าเดินถอยหลัง
```

```
for i in range(5, 0, -1):
```

```
    print(i)
```

```
↳ 5
```

```
4
```

```
3
```

```
2
```

```
1
```

แบบฝึกหัด 7.2

1. จงวนลูปค่า index i จากค่า 1 ถึง 5 แล้วพิมพ์ค่าของ $2 * i$ ออกมาทางหน้าจอ

เขียนโค้ด...



2

4

6

8

2. จงวนรูปค่า index i จากค่า 20 จนถึง 15 แล้วพิมพ์ค่า $i / 10$ ออกมาทางหน้าจอ

เขียนโค้ด...

↩ 2.0

1.9

1.8

1.7

1.6

3, จงวนรูปเพื่อแสดงสูตรคูณแม่ 3 ไล่
ตั้งแต่ 3×1 จนถึง 3×12 โดยจะได้ผลลัพธ์
ดังนี้

$$3 * 1 = 3$$

$$3 * 2 = 6$$

$$3 * 3 = 9$$

:

:

:

$$3 * 12 = 36$$

เขียนโค้ด...



$$3 * 1 = 3$$

$$3 * 2 = 6$$

$$3 * 3 = 9$$

$$3 * 4 = 12$$

$$3 * 5 = 15$$

$$3 * 6 = 18$$

$$3 * 7 = 21$$

$$3 * 8 = 24$$

$$3 * 9 = 27$$

$$3 * 10 = 30$$

$$3 * 11 = 33$$

$$3 * 12 = 36$$

ลูปซ้อนลูป (loop embedding)

เราสามารถเขียนลูปซ้อนลูปได้ด้วย หากว่าโปรแกรมของเรามีความซับซ้อนกว่าลูปเดียว เช่น เราสามารถวนลูปเพื่อไล่สูตรคูณแม่ 3 ถึงแม่ 5 ได้ดังนี้

```
# วนลูป index m เพื่อไล่แม่สูตรคูณจากแม่ 3 ถึงแม่ 5
for m in range(3, 6):
    print('Multiplication of', m)
    # วนลูป index i เพื่อไล่ตัวคูณจาก 1 ถึง 12
    for i in range(1, 13):
        print(m, '*', i, '=', m * i)
    # พิมพ์บรรทัดใหม่คั่นระหว่างแม่
    print()
```

Output:

```
↳ Multiplication of 3
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30
3 * 11 = 33
3 * 12 = 36
```

```
Multiplication of 4
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
4 * 9 = 36
4 * 10 = 40
4 * 11 = 44
4 * 12 = 48
```

```
Multiplication of 5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
5 * 11 = 55
5 * 12 = 60
```

ข้อสังเกตจากการวนลูปฟอรัในการไล่แม่สูตรคูณ ก็คือ

- 1) ในลูปนอก เราจะวนลูป index ของแม่สูตรคูณก่อน เช่น แม่ 3 ถึงแม่ 5 เป็นต้น
- 2) ส่วนลูปข้างใน เราจะวนลูป index ของตัวคูณในแต่ละแม่ คือตั้งแต่ 1 ถึง 12

นั่นแสดงว่า ถ้าเราจะเขียนโปรแกรมที่สลับซับซ้อนได้ เราจะต้องแยกแยะ และถอดลูปออกเป็นชั้นนอกและชั้นในให้ออก

แบบฝึกหัด 7.3

1. จงเขียนโปรแกรมเพื่อไล่แม่สูตรคูณจากแม่ 11 ถึงแม่ 15 โดยแต่ละแม่จะคูณกับตัวเลขตั้งแต่ 1 จนถึง 12

เขียนโค้ด...



กิจกรรม 8: การกำหนดเงื่อนไข

การกำหนดเงื่อนไข (branching) ในภาษาไพธอนจะใช้คำสั่ง if (แปลว่า 'ถ้า') โดยจะมีรูปแบบดังนี้
if <เงื่อนไข>:

<กระบวนการ>

คอมไพเลอร์จะทำกระบวนการที่อยู่ในด้านใน ถ้าหากว่าเงื่อนไขที่กำหนดนี้เป็นจริง

จะสังเกตว่ากระบวนการนี้จะต้องย่อหน้าเหมือนกับคำสั่งลูป for นั่นเป็นเพราะว่า การเว้นวรรคย่อหน้าจะเป็นการบอกสโคป (scope) ของคำสั่ง ว่าเป็นกระบวนการที่อยู่ภายใต้คำสั่งที่อยู่ด้านนอก การวนลูปก็จะวนภายใน scope ของตัวเอง การกำหนดเงื่อนไขก็จะทำงานเฉพาะภายใน scope ของตัวเองเท่านั้น

เช่น โปรแกรมด้านล่างนี้จะพิมพ์ข้อความ 'Hurray!' ถ้าหากว่าเงินเดือนในตัวแปร salary มีค่ามากกว่า 10,000 บาท

ทดลองพิมพ์และรัน

```
salary = 5000  
if salary > 10000:  
    print('Hurray!')
```

↳ No Output

ทดลองเปลี่ยนค่าตัวเลขของตัวแปร salary เป็น 12000 แล้วรันใหม่

↳ Hurray!

รูปแบบของเงื่อนไข

เราสามารถใช้อุปกรณ์เปรียบเทียบทางคณิตศาสตร์ได้เหมือนในวิชาคณิตศาสตร์

เครื่องหมาย	ความหมาย	ตัวอย่าง
<	น้อยกว่า	$3 < 5$
>	มากกว่า	$5 > 3$
==	เท่ากับ	<code>my_salary == your_salary</code>
!=	ไม่เท่ากับ	<code>my_salary != your_salary</code>
>=	มากกว่าหรือเท่ากับ	<code>my_salary >= your_salary</code>
<=	น้อยกว่าหรือเท่ากับ	<code>my_salary <= your_salary</code>

ถ้าเราลองพิมพ์ค่าความจริงของเงื่อนไขออกมา เราจะได้เป็นค่า True หรือ False ซึ่งค่าความจริงก็เป็นข้อมูลประเภทหนึ่งเหมือนกัน เรียกว่า *ค่าตรรกะ* (boolean) เช่น

ทดลองพิมพ์และรัน

```
print(0.5 > 0.3)      # True
```

```
print(0.25 < -1.0)   # False
```

```
print(1.25 == 1.25)  # True
```

```
print(1.30 != -0.78) # True
```

↪ True

False

True

True

ทั้งนี้เราสามารถใช้เครื่องหมายเปรียบเทียบสามารถใช้ได้กับข้อความได้ เช่น

ทดลองพิมพ์และรัน

```
print('hello' == 'hello')    # True
```

```
print('hello' != 'world')    # True
```

```
print('hello' < 'alphabet')  # False – คอมพิวเตอร์เรียงลำดับข้อความตามตัวอักษร
```

```
↳ True
```

```
True
```

```
False
```

นอกจากนี้เรายังสามารถใช้ตัวเชื่อมทางตรรกศาสตร์ได้ด้วย

ตัวเชื่อม	ความหมาย	ตัวอย่าง
and	และ	$a == b$ and $x == y$
or	หรือ	$a != b$ or $c != d$
not	ไม่	not ($a > b$)
()	วงเล็บ	$(a == b)$ and not ($c > d$)

ตารางค่าความจริง

ตรรกศาสตร์

ตารางค่าความจริง

P	q	และ $P \wedge q$	หรือ $P \vee q$	ถ้า...แล้ว $P \rightarrow q$	ก็ต่อเมื่อ $P \leftrightarrow q$	นิเสธ $\sim P$
T	T	T	T	T	T	F
T	F	F	T	F	F	F
F	T	F	T	T	F	T
F	F	F	F	T	T	T

ทดลองพิมพ์และรัน

```
v1 = 100
```

```
v2 = 200
```

```
v3 = 300
```

```
print(v1 == v2 and v2 == v3)
```

```
# False and False = False
```

```
print(v1 + v2 == v3 or v2 + v3 == v1)
```

```
# True or False = True
```

```
print(not(v1 == v2 and v2 == v3) or v1 == v2)
```

```
# not(False and False) or False = True
```

```
↪ False
```

```
True
```

```
True
```

แบบฝึกหัด 8.1

จงตั้งเงื่อนไขเพื่อให้คอมพิวเตอร์พิมพ์ข้อความว่า 'You are tall.'

ถ้าหากว่าความสูงในตัวแปร `my_height` มากกว่า 170 กำหนดให้ตัวแปร `my_height` มีค่าเท่ากับ 175

เขียนโค้ด...

↳ You are tall.

แบบฝึกหัด 8.2

จงตั้งเงื่อนไขเพื่อให้คอมพิวเตอร์พิมพ์ข้อความว่า 'You are weathy and tall.' ถ้าหากว่าเงินเดือนในตัวแปร `my_salary` มากกว่า 10000 และความสูงในตัวแปร `my_height` มากกว่า 170

เขียนโค้ด...

↳ You are wealthy and tall.

แบบฝึกหัด 8.3

จงตั้งเงื่อนไขเพื่อควบคุมการพิมพ์ความสูงของนักเรียนแต่ละคน โดยจะพิมพ์ออกมาเฉพาะคนที่มีความสูงมากกว่า 160 เซนติเมตรเท่านั้น [หมายเหตุ: จะสังเกตว่าโจทย์ข้อนี้จะใช้ทั้งลูปฟอร์และการตั้งเงื่อนไขประกอบกันได้ด้วย]

```
student_heights = [163.5, 150.0, 167.0, 161.25, 170.0]
```

เขียนโค้ด...

↳ 163.5

167.0

161.25

170.0

กิจกรรม 9: ฟังก์ชัน

การสร้างฟังก์ชันใช้เองใน Python แบบใช้ **def** keyword เช่น

```
def function_name( ):  
    do something
```

ตัวอย่าง เช่น

สร้าง function

```
def sum_two_nums(a, b):  
    return a+b
```

ทดสอบเรียกใช้ function

```
sum_two_nums(6, 4)
```

↪ 10

```
sum_two_nums(10, 2) #
```

↪ 12

Lambda Functions

- ฟังก์ชันที่ไม่มีการตั้งชื่อ
- ทำงานเหมือนฟังก์ชันธรรมดา
- เขียนโค้ดสั้นกว่า กระชับกว่า
- ข้อจำกัด คือ มีได้เพียงหนึ่งคำสั่ง (Expression)
- ใช้แทนฟังก์ชันง่าย ๆ ไม่ซับซ้อน

รูปแบบ Lambda Functions

lambda argument : expression

ตัวอย่างเช่น

```
cube = lambda x: x**3  
print(cube(2))
```

เปรียบเทียบฟังก์ชันธรรมดา กับ ฟังก์ชัน Lambda

ฟังก์ชัน Lambda

```
cube = lambda x: x**3
```

```
print(cube(2))
```

ฟังก์ชันธรรมดา

```
def cube2(y)  
    result = y**3  
    return result
```

```
print(cube2(3))
```

แบบฝึกหัด 9.1

1. เขียนฟังก์ชันธรรมดา รับค่า 4 พารามิเตอร์ และหาค่าเฉลี่ย
2. เขียนฟังก์ชันไม่มีชื่อให้รับพารามิเตอร์ 1 ตัว และป้อนค่านับหารด้วย 3

Reference

- ธนารักษ์ ธีระมั่นคง และคณะ. (2562). การโปรแกรมภาษา Python (ไพธอน) เบื้องต้น. กรุงเทพฯ: สมาคมปัญญาประดิษฐ์ประเทศไทย.
- กษิติศ สตางค์มงคล (2563). Python for Non-Programmer. กรุงเทพฯ: Datarockie.
- กฤษฎา เฉลิมสุข (2563). Introduction to Python for Data Science. กรุงเทพฯ: The Self Made Serial Entrepreneur.
- <http://www.ayarafun.com>