บทที่ 5

โครงสร้างข้อมูลภาษาโปรแกรมไพธอน

ในบทนี้เราจะได้รู้จักกับโครงสร้างข้อมูลภาษาโปรแกรมไพธอน (Python Data Structure) ซึ่งประกอบด้วยข้อมูลชนิดลิสต์ (List) ข้อมูลชนิดทูเพิล (Tuple) ข้อมูลชนิดดิกชันนารี (Dictionary) และข้อมูลชนิดเซต (Set) ที่ใช้จัดเก็บข้อมูลแบบลำดับ (Sequence) โดยข้อมูลชนิดเหล่านี้สามารถเก็บข้อมูลต่างประเภทกันได้ ทั้งข้อมูลชนิดจำนวนเต็ม และข้อมูลชนิดสตริง นอกจากนี้ยังสามารถที่จะเก็บข้อมูลชนิดต่างชนิดซ้อนกัน เช่น ลิสต์ซ้อนลิสต์ ทูเพิลซ้อนทูเพิล หรือลิสต์ซ้อนทูเพิลก็ได้ แต่ข้อมูลชนิดเหล่านี้อาจจะมีความแตกต่างกันในการเพิ่ม ลบ แก้ไขข้อมูล ซึ่งผู้อ่านจะได้ศึกษาในบทนี้

5.1 รู้จักกับข้อมูลชนิดลิสต์

ข้อมูลชนิดลิสต์ (List) คล้ายกับข้อมูลอาร์เรย์ในภาษาอื่น ๆ การประกาศสร้างตัวแปรขึ้นมาเก็บข้อมูลชนิดลิสต์ไม่แตกต่างจากการสร้างตัวแปรข้อมูลชนิดอื่น ๆ แต่จะใช้เครื่องหมาย `[...]` จัดเก็บข้อมูล และใช้เครื่องหมาย Comma (`,`) คั่นระหว่างสมาชิกในลิสต์ หากเป็นข้อมูลชนิดสตริงให้ใส่เครื่องหมาย `'...'` หรือ `"..."` ครอบ นอกจากนี้สมาชิกใน List จะสามารถประกอบด้วยตัวแปรหลายชนิดรวมกันได้ เช่น จำนวนเต็ม, ทศนิยม, สตริง พิจารณาจากตัวอย่างต่อไปนี้

ตัวอย่าง 5.1
การประกาศตัวแปรข้อมูลชนิดลิสต์

```
1  books_lst = ['Python', 'Java', 'C', 'C++', 'C#']
2  # สร้างตัวแปรข้อมูลชนิดลิสต์มีสมาชิกเป็นข้อมูลชนิดสตริง
3  numbers_lst = [1, 2, 3, 4, 5, 6]
4  # สร้างตัวแปรข้อมูลชนิดลิสต์มีสมาชิกเป็นข้อมูลชนิดจำนวนเต็ม
5  book_num_lst = [1, 'Python', 2, 'Java', 3 ,'C']
6  # สร้างตัวแปรข้อมูลชนิดลิสต์มีสมาชิกเป็นข้อมูลชนิดจำนวนเต็มและสตริง
7  print('รายชื่อหนังสือใน books_lst = ', books_lst)
8  # แสดงผลลัพธ์ตัวแปร book_lst
9  print('สมาชิกใน numbers_lst =', numbers_lst)
10 # แสดงผลลัพธ์ตัวแปร numbers_lst
11 print('สมาชิกใน book_num_lst =', book_num_lst)
12 # แสดงผลลัพธ์ตัวแปร book_num_lst
```

```
รายชื่อหนังสือใน books_1st = ['Python', 'Java', 'C', 'C++','C#']
สมาชิกใน numbers_1st = [1, 2,3,4,5,6]
สมาชิกใน book_num_1st = [1, 'Python', 2, 'Java' , 3, 'C']
```

5.1.1 การเข้าถึงตำแหน่งข้อมูลของข้อมูลชนิดลิสต์

เราสามารถเข้าถึงตำแหน่งข้อมูลที่เก็บอยู่ภายในข้อมูลชนิดลิสต์ได้สองทางคือ จากทางด้านหน้าและจากทางด้านท้ายสุด ด้วยการระบุตำแหน่ง (Index) หากเข้าถึงข้อมูลจากทางด้านหน้าให้เริ่มนับจากตำแหน่งที่ 0 ถ้าเข้าถึงตำแหน่งข้อมูลจากทางด้านท้ายสุดจะเริ่มนับจากตำแหน่งที่ -1 มีรูปแบบการเข้าถึงตำแหน่งดังนี้

```
var[start : stop : step]
```

`start`	ออบเจ็คที่ต้องการแสดงผล มีได้มากกว่า 1 ออบเจ็ค
`stop`	ส่วนที่ใช้แยกแต่ละออบเจ็คออกจากกัน ค่าปกติเป็นช่องว่าง
`step`	การกำหนดแสดงผลออบเจ็ค ค่าปกติเป็นการขึ้นบรรทัดใหม่

ตัวอย่าง 5.2
การเขียนคำสั่งโปรแกรมระบุตำแหน่งข้อมูลที่เก็บอยู่ในข้อมูลชนิดลิสต์

```
1 books_lst = ['Python', 'Java', 'C', 'C++', 'C#', 'Scala', 'PHP']
2 print('ตำแหน่ง -5 ใน books_lst คือ', books_lst[-5])
3 print('ตำแหน่ง 2-6 ใน books_lst คือ', books_lst[2:5])
4 print('ตำแหน่ง 0-3 ใน books_lst คือ', books_lst[:4])
5 print('แสดงข้อมูลตำแหน่งที่ 1-5 โดยข้ามครั้งละ 2 ตำแหน่ง=' ,
  ↪  books_lst[1:6:2])
6 print('แสดงข้อมูลใน books_lst โดยข้ามครั้งละ 2 ตำแหน่ง= ', books_lst[::2])
```

```
ตำแหน่ง -5 ใน books_lst คือ C
ตำแหน่ง 2-6 ใน books_lst คือ ['C', 'C++', 'C#']
ตำแหน่ง 0-3 ใน books_lst คือ ['Python', 'Java', 'C', 'C++']
แสดงข้อมูลตำแหน่งที่ 1-5 โดยข้ามครั้งละ 2 ตำแหน่ง= ['Java', 'C++', 'Scala']
แสดงข้อมูลใน books_lst โดยข้ามครั้งละ 2 ตำแหน่ง=  ['Python', 'C', 'C#',
↪  'PHP']
```

จากตัวอย่างโปรแกรมได้แสดงวิธีการเขียนคำสั่งโปรแกรม การเข้าถึงตำแหน่งข้อมูลในข้อมูลชนิดลิสต์ ผู้อ่านจะสังเกตเห็นว่าไม่จำเป็นต้องระบุตำแหน่งก็ได้ เมื่อต้องการเข้าถึงจากตำแหน่งเริ่มต้นจนถึงตำแหน่งสุดท้าย แต่ต้องคั่นด้วยเครื่องหมาย `colon(:)`

5.1.2 เมธอดและฟังก์ชันที่ใช้กับข้อมูลชนิดลิสต์

ข้อมูลชนิดลิสต์มีเมธอดให้เรียกใช้งานจำนวนมาก หากผู้อ่านต้องการรายละเอียดเพิ่มเกี่ยวกับการใช้งาน สามารถใช้ `help(list)` เพื่อขอดูตัวอย่างการใช้งานได้

เมธอด	ความหมาย	รูปแบบการใช้งาน
`append()`	เพิ่มข้อมูลต่อท้ายลิสต์	`lst.appened(x)` `x` คือ ข้อมูลที่ต้องการเพิ่ม
`clear()`	ลบข้อมูลทั้งหมดออกจากลิสต์	`lst.clear()`
`copy()`	คัดลอกข้อมูลชนิดลิสต์	`lst_new = lst.copy()` `lst_new` คือ ตัวแปรเก็บข้อมูลชนิดลิสต์ `lst` คือ ตัวแปรข้อมูลชนิดลิสต์ที่ถูกคัดลอก
`count()`	นับจำนวนข้อมูลที่ซ้ำกันในลิสต์	`lst.count(x)` `x` คือ ข้อมูลที่ต้องการนับ
`extend()`	เพิ่มลิสต์เข้าไปในลิสต์	`lst.extend(x)` `x` คือ ลิสต์ที่ต้องการเพิ่ม
`index()`	ค้นหาตำแหน่งข้อมูลที่เก็บอยู่ในลิสต์ ถ้าไม่มีตำแหน่งระบุจะแจ้งเตือนข้อผิดพลาด	`lst.index(x, start, stop)` `x` คือ ข้อมูลที่ต้องการค้นหา `start` คือ จุดเริ่มที่ต้องการค้นหา `stop` คือ จุดสุดท้ายที่ต้องการให้ค้นหา
`insert()`	แทรกข้อมูลเข้าไปในลิสต์โดยการระบุตำแหน่ง	`lst.insert(index, x)` `index` คือ ตำแหน่งที่ต้องการแทรก `x` คือ ข้อมูลที่ต้องการแทรก

ตาราง 5.1: เมธอดและฟังก์ชันที่ใช้กับข้อมูลชนิดลิสต์ (มีต่อ)

เมธอด	ความหมาย	รูปแบบการใช้งาน
`pop()`	ลบข้อมูลโดยการระบุตำแหน่ง	`lst.pop(index)` `index` คือ ตำแหน่งที่ต้องการลบ
`remove()`	ลบข้อมูลด้วยการระบุชื่อ ถ้าไม่มีข้อมูลที่ระบุจะแจ้งเตือนข้อผิดพลาด	`lst.remove(x)` `x` คือ ชื่อข้อมูลที่ต้องการลบ
`reverse()`	สลับตำแหน่งข้อมูลจากด้านหลังมาด้านหน้า	`lst.reverse()`
`sort()`	เรียงลำดับข้อมูลที่อยู่ในลิสต์	`lst.sort(reverse=False)` `reverse=False` คือ ค่าเริ่มต้นจะเรียงจากมากไปหาน้อย กำหนดหรือไม่กำหนดก็ได้ ถ้ากำหนดให้เป็น `True` จะเรียงจากมากไปหาน้อย
`len()`	ฟังก์ชันที่ใช้แสดงจำนวนข้อมูลที่มีอยู่ในลิสต์	`len(lst)`

ตาราง 5.1: เมธอดและฟังก์ชันที่ใช้กับข้อมูลชนิดลิสต์

ตัวอย่าง 5.3
การเขียนคำสั่งโปรแกรมการใช้เมธอดและฟังก์ชันกับข้อมูลชนิดลิสต์

```
1 books = ['Python', 'Java', 'C', 'C++', 'C#']
2 numbers = [1, 50, 4, 51, 4, 6, 10, 15]
3
4 books.append('Julia')
5 print('แสดงรายการหนังสือใน books = ', books)
6
7 in_book = books.index('C', 1, 5)
```

```
print('ตำแหน่งหนังสือภาษา C ใน books = ', in_book)

books.pop(1)
print('รายชื่อที่มีอยู่ใน books = ', books)

books.reverse()
print('สลับข้อมูลจากด้านหลังมาด้านหน้า = ', books)

books.sort()
print('เรียงรายชื่อหนังสือใน books = ', books)

books.sort(reverse=True)
print('เรียงรายชื่อหนังสือใน books = ', books)

print('จำนวนข้อมูลใน numbers = ', len(numbers))
```

```
แสดงรายการหนังสือใน books =  ['Python', 'Java', 'C', 'C++', 'C#',
↪  'Julia']
ตำแหน่งหนังสือภาษา C ใน books =  2
รายชื่อที่มีอยู่ใน books =  ['Python', 'C', 'C++', 'C#', 'Julia']
สลับข้อมูลจากด้านหลังมาด้านหน้า =  ['Julia', 'C#', 'C++', 'C', 'Python']
เรียงรายชื่อหนังสือใน books =  ['C', 'C#', 'C++', 'Julia', 'Python']
เรียงรายชื่อหนังสือใน books =  ['Python', 'Julia', 'C++', 'C#', 'C']
จำนวนข้อมูลใน numbers =  8
```

5.1.3 การดำเนินการกับข้อมูลชนิดลิสต์

เราสามารถดำเนินการต่าง ๆ กับข้อมูลชนิดลิสต์ได้ เช่น การทำซ้ำลิสต์ การเชื่อมลิสต์เข้าด้วยกัน การเปรียบเทียบลิสต์ เป็นต้น แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 5.4
การเขียนคำสั่งโปรแกรมการดำเนินการต่าง ๆ กับข้อมูลชนิดลิสต์

```
# สร้างตัวแปรข้อมูลชนิดลิสต์เป็นข้อมูลชนิดจำนวนเต็ม
nums1 = [4, 5, 7, 15]
nums2 = [4, 3, 5, 78]

# สร้างตัวแปรข้อมูลชนิดลิสต์เป็นข้อมูลชนิดสตริง
books = ['Python', 'Java', 'C', 'C++','C#', 'Scala' ]

# แสดงผลลัพธ์การเชื่อมต่อค่าตัวแปร nums กับ books
print (nums1 + books)
# แสดงผลลัพธ์การทำซ้ำค่าตัวแปร nums ทั้งหมด 3 ครั้ง
print(nums1 * 3)
# แสดงผลลัพธ์การเปรียบเทียบค่าตัวแปร nums กับ num_lst1
print('num_lst กับ nums1 เท่ากันหรือไม่ = ', nums1 == nums2)

# นำค่าตัวแปร nums และ book_lst เก็บไว้ในตัวแปร news_lst
news_lst = [nums1, books]
# แสดงผลลัพธ์ค่าตัวแปร news_lst
print(news_lst)
# แสดงผลลัพธ์การตรวจสอบ Python เป็นสมาชิกของ books?
print('Python เป็นสมาชิกของ books_list =', 'Python' in books)
```

```
[4, 5, 7, 15, 'Python', 'Java', 'C', 'C++', 'C#', 'Scala']
[4, 5, 7, 15, 4, 5, 7, 15, 4, 5, 7, 15]
num_lst กับ nums1 เท่ากันหรือไม่ =  False
[[4, 5, 7, 15], ['Python', 'Java', 'C', 'C++', 'C#', 'Scala']]
Python เป็นสมาชิกของ books_list = True
```

เราสามารถใช้ฟังก์ชัน `min()`, `max()` และ `sum()` ดำเนินการกับสมาชิกที่เก็บไว้ในตัวแปรข้อมูลชนิดลิสต์ที่เป็นข้อมูลชนิดจำนวนเต็มหรือจำนวนทศนิยมได้ โดยที่ฟังก์ชัน `min()` ใช้ค้นหาค่าข้อมูลที่น้อยที่สุดที่เก็บอยู่ในลิสต์ ฟังก์ชัน `max()` ใช้หาค่าที่มากที่สุดที่เก็บอยู่ในลิสต์ และฟังก์ชัน `sum()` ใช้สำหรับรวมค่าข้อมูลทั้งหมดที่มีอยู่ในลิสต์ ดังตัวอย่างต่อไปนี้

ตัวอย่าง 5.5
การเขียนคำสั่งโปรแกรมการดำเนินการต่าง ๆ กับข้อมูลชนิดลิสต์

```
# สร้างตัวแปรข้อมูลชนิดลิสต์เป็นข้อมูลชนิดจำนวนเต็ม
numbers = [1, 4, 5, 6, 15]
# แสดงผลลัพธ์การหาค่าที่น้อยที่สุดด้วยฟังก์ชัน min()
print('ค่าที่น้อยที่สุดใน numbers =', min(numbers))
# แสดงผลลัพธ์การหาค่าที่มากที่สุดด้วยฟังก์ชัน max()
print('ค่าที่มากที่สุดใน numbers = ', max(numbers))
# แสดงผลลัพธ์การหาค่าผลรวมด้วยฟังก์ชัน sum()
print('ผลรวมของ numbers = ', sum(numbers))
```

```
ค่าที่น้อยที่สุดใน numbers = 1
ค่าที่มากที่สุดใน numbers =  15
ผลรวมของ numbers =  31
```

5.2 รู้จักกับข้อมูลชนิดทูเพิล

ข้อมูลชนิดทูเพิล (Tuple) มีลักษณะการจัดเก็บข้อมูลเหมือนข้อมูลชนิดลิสต์คือ จัดเก็บข้อมูลแบบลำดับและสมาชิกที่เก็บอยู่ภายในข้อมูลชนิดทูเพิลเป็นแบบใดก็ได้ แต่ข้อมูลชนิดทูเพิลไม่สามารถแก้ไขได้ การสร้างข้อมูลชนิดทูเพิลขึ้นมาใช้งาน สมาชิกทั้งหมดจะอยู่ภายในเครื่องหมายวงเล็บ **(. . .)** และจะถูกคั่นด้วยเครื่องหมาย Comma **(,)**

ตัวอย่าง 5.6
การเขียนคำสั่งโปรแกรมสร้างข้อมูลชนิดทูเพิลขึ้นมาใช้งาน

```
# สร้างตัวแปรข้อมูลชนิดทูเพิลที่เป็นค่าว่าง
tup = ()
# สร้างตัวแปรเป็นข้อมูลชนิดทูเพิลที่มีสมาชิกเป็นข้อมูลชนิดสตริง ลิสต์ และทูเพิล
var_tup = ('Name', 'Address', [1, 2, 3], ('ID', 3451))
# สร้างตัวแปรเป็นข้อมูลชนิดสตริง
book = ('Python')
# สร้างตัวแปรเป็นข้อมูลชนิดทูเพิล มีเครื่องหมาย (,)
book_tup = ('Python', 'C++')

# แสดงผลลัพธ์จากค่าตัวแปร var_tup
print(var_tup)
# แสดงผลลัพธ์ข้อมูลชนิดของค่าตัวแปร var_tup
print('ข้อมูลชนิดของ var_tup คือ', type(var_tup))
# แสดงผลลัพธ์ข้อมูลชนิดของค่าตัวแปร book
print('ข้อมูลชนิดของ book คือ', type(book))
# แสดงผลลัพธ์ข้อมูลชนิดของค่าตัวแปร book_tup
print('ข้อมูลชนิดของ book_tup คือ', type(book_tup))
```

```
('Name', 'Address', [1, 2, 3], ('ID', 3451))
ข้อมูลชนิดของ var_tup คือ <class 'tuple'>
ข้อมูลชนิดของ book คือ <class 'str'>
ข้อมูลชนิดของ book_tup คือ <class 'tuple'>
```

จากที่ได้กล่าวมาแล้วว่าข้อมูลชนิดทูเพิลไม่สามารถดำเนินการแก้ไข หากเราพยายามกระทำการดังกล่าว จะมีการแจ้งเตือนข้อผิดพลาดเกิดขึ้น แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 5.7
การแจ้งเตือนข้อผิดพลาดเมื่อทำการแก้ไขข้อมูลชนิดทูเพิล

```
1 books = ('Python', 'Java', 'C','C++', 'C#')
2 print(books)
3 print(books[0])
4 books[0] = 'PYTHON'      # ให้ค่า books[0] ใหม่เป็น 'PYTHON'
5 print(books)
```

```
('Python', 'Java', 'C', 'C++', 'C#')
Python
Traceback (most recent call last):
  File "src/test.py", line 4, in <module>
    books[0] = 'PYTHON'      # ให้ค่า books[0] ใหม่เป็น 'PYTHON'
TypeError: 'tuple' object does not support item assignment
```

จากตัวอย่าง 5.7 ตัวแปร `books` เป็นตัวแปรชนิดทูเพิลที่มีสมาชิกเป็นสตริงจำนวน 4 ตัว และในบรรทัดที่ 4 เป็นคำสั่งโปรแกรมให้ค่าตัวแปร `books` ในตำแหน่งแหน่งแรก (`books[0]`) แต่ทำให้เกิดการแจ้งเตือนข้อผิดพลาด เนื่องจากข้อมูลชนิดทูเพิลไม่อนุญาตรองรับการให้ค่าสมาชิก (Item Assignment)

5.2.1 เมธอดและฟังก์ชันที่ใช้กับข้อมูลชนิดทูเพิล

ข้อมูลชนิดทูเพิลเก็บข้อมูลชนิดได้หลากหลายประเภท เช่น จำนวนเต็ม ทศนิยม สตริง ลิสต์ เป็นต้น ภาษาโปรแกรมไพธอนได้จัดเตรียมเมธอดต่าง ๆ และฟังก์ชันไว้ให้เรียกใช้งานดังต่อไปนี้

เมธอด	ความหมาย	รูปแบบการใช้งาน
`index()`	ใช้สำหรับแสดงตำแหน่งข้อมูล	`tup.index(x)`
`count()`	ใช้สำหรับนับจำนวนข้อมูล	`tup.count(x)`
`len()`	ฟังก์ชันที่ใช้แสดงจำนวนข้อมูลที่มีอยู่ในทูเพิล	`len(tup)`

ตาราง 5.2: เมธอดและฟังก์ชันในข้อมูลชนิดทูเพิล เมื่อ `tup` คือ ตัวแปรข้อมูลชนิดทูเพิล `x` คือ ค่าข้อมูลใด ๆ

ตัวอย่าง 5.8
การเขียนคำสั่งโปรแกรมการใช้เมธอดและฟังก์ชันกับข้อมูลชนิดทูเพิล

```
# สร้างตัวแปรข้อมูลชนิดทูเพิล
books = ('Python', 'Java', 'C', 'C++', 'C#', 'PHP')
# แสดงผลลัพธ์ตำแหน่งด้วยเมธอด index()
print('ตำแหน่งของหนังสือภาษา Python คือ ', books.index('Python'))
# นับรายการข้อมูลที่ต้องการด้วยเมธอด count()
print('จำนวนหนังสือภาษา Python ทั้งหมด คือ', books.count('Python'),
    'เล่ม')
# แสดงผลลัพธ์รายการข้อมูลทั้งหมดด้วยฟังก์ชัน len()
print('จำนวนหนังสือทั้งหมดใน books คือ', len(books), 'เล่ม')
```

```
ตำแหน่งของหนังสือภาษา Python คือ  0
จำนวนหนังสือภาษา Python ทั้งหมด คือ 1 เล่ม
จำนวนหนังสือทั้งหมดใน books คือ 6 เล่ม
```

5.2.2 การดำเนินการกับข้อมูลชนิดทูเพิล

เราสามารถดำเนินการต่าง ๆ กับข้อมูลชนิดทูเพิลเหมือนกับข้อมูลชนิดลิสต์ เช่น การเปรียบเทียบ การทำซ้ำ การตรวจสอบสมาชิก การหาค่าสูงสุด การหาค่าต่ำสุด การหาค่าผลรวม เป็นต้น แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 5.9
การเขียนคำสั่งโปรแกรมตรวจสอบการเป็นสมาชิกในข้อมูลชนิดทูเพิล

```
# สร้างตัวแปรข้อมูลชนิดทูเพิลเป็นข้อมูลชนิดสตริง
books_tup = ('Python', 'Java', 'C', 'C++', 'C#')
# แสดงผลลัพธ์การตรวจสอบ ".Net" ในค่าตัวแปร books_tup
print('มีหนังสือ SQL หรือไม่ = ', 'SQL' in books_tup)
# แสดงผลลัพธ์การตรวจสอบ "C#" ในค่าตัวแปร books_tup
print('มีหนังสือ C++ หรือไม่ = ', 'C++' in books_tup)
```

```
มีหนังสือ SQL หรือไม่ =  False
มีหนังสือ C++ หรือไม่ =  True
```

ตัวอย่าง 5.10
การเขียนคำสั่งโปรแกรมหาค่าต่ำสุด ค่าสูงสุด และผลรวม จากข้อมูลชนิดทูเพิล

```
# สร้างตัวแปรข้อมูลชนิดทูเพิลเป็นข้อมูลชนิดจำนวนเต็ม
numbers = (4, 5, 4.12, 7, 15)
# แสดงผลลัพธ์การหาค่าที่มากที่สุดด้วยฟังก์ชัน max()
print ('ค่าที่มากที่สุดใน numbers คือ', max(numbers))
# แสดงผลลัพธ์การหาค่าที่ต่ำที่สุดด้วยฟังก์ชัน min()
print ('ค่าที่ต่ำที่สุดใน numbers คือ', min(numbers))
# แสดงผลลัพธ์การหาค่าผลรวมด้วยฟังก์ชัน sum()
print ('ผลรวมของ numbers คือ', sum(numbers))
```

```
ค่าที่มากที่สุดใน numbers คือ 15
ค่าที่ต่ำที่สุดใน numbers คือ 4
ผลรวมของ numbers คือ 35.120000000000005
```

ตัวอย่าง 5.11
การเปรียบเทียบ การเชื่อม การทำซ้ำ ข้อมูลชนิดทูเพิล

```
n_1 = (-1, -2, -1, 0, 12)
n_2 = (1, 2, 3.14, 10)
n_3 = (1, 2, 3.14, 10)
books_1 = ('Python', 'Java', 'C', 'C++', 'SQL')
books_2 = ('C', 'C++', 'C#')
print('ผลเปรียบเทียบ n_1 < n_3 คือ', n_1 < n_3)
print('ผลเปรียบเทียบ n_2 == n_3 คือ', n_2 == n_3)
print('ผลการรวมระหว่าง books_1 กับ booksle_2 คือ', books_1 + books_2)
print('books_2 * 2 คือ', books_2 * 2)
```

```
ผลเปรียบเทียบ n_1 < n_3 คือ True
ผลเปรียบเทียบ n_2 == n_3 คือ True
ผลการรวมระหว่าง books_1 กับ booksle_2 คือ ('Python', 'Java', 'C', 'C+
↪  +', 'SQL', 'C', 'C++', 'C#')
books_2 * 2 คือ ('C', 'C++', 'C#', 'C', 'C++', 'C#')
```

5.3 รู้จักกับข้อมูลชนิดดิกชันนารี

ข้อมูลดิกชันนารี (Dictionary) สามารถเปลี่ยนแปลงข้อมูลได้เหมือนกับข้อมูลชนิดลิสต์ คือ สามารถเพิ่ม ลบ แก้ไขข้อมูล แต่มีความแตกต่างระหว่างข้อมูลชนิด 2 ประเภทนี้คือ ข้อมูลชนิดดิกชันนารีจะเก็บข้อมูลแบบคู่ประกอบด้วย คีย์ (Key) และค่าข้อมูล (Value) การจัดเก็บข้อมูลจะอยู่ในเครื่องหมายวงเล็กปีกกา `{...}` แยกส่วนของ Key และ Value ด้วยเครื่องหมาย Colon (`:`) และใช้เครื่องหมาย Comma (`,`) เพื่อแยกแต่ละคู่ Key และ Value ตัวอย่างวิธีการเขียนคำสั่งโปรแกรมสร้างชนิดข้อมูลดิกชันนารีมีดังนี้

โครงสร้างข้อมูลชนิดดิกชันนารี

```
var = {key_1: val_1, ..., key_n: val_n}
```

`key_1, ..., key_n`	ข้อมูลแบบเปลี่ยนค่าไม่ได้ (Immutable) เช่นข้อมูลชนิดจำนวน สตริง หรือทูเพิล โดยที่ค่าทั้งหมดต้องไม่ซ้ำกัน
`val_1, ...., val_n`	ข้อมูลชนิดใดก็ได้

ตัวอย่าง 5.12
การเขียนคำสั่งโปรแกรมสร้างข้อมูลชนิดดิกชันนารี

```
# Key เป็นข้อมูลชนิดจำนวนเต็ม และ Value เป็นข้อมูลชนิดสตริง
animals = {1: 'Cat', 2: 'Dog', 3: 'Tiger', 4: 'Bird', 5: 'Lion'}

# Key และ Value เป็นข้อมูลชนิดสตริง
sports = {'1': 'Football', '2': 'Tennis', '3': '', '4':
    'Runnig'}

print(animals)
print(sports)
```

```
{1: 'Cat', 2: 'Dog', 3: 'Tiger', 4: 'Bird', 5: 'Lion'}
{'1': 'Football', '2': 'Tennis', '3': '', '4': 'Runnig'}
```

ในส่วนของ Value สามารถเป็นได้ทั้งข้อมูลชนิดสตริง จำนวนเต็ม ทศนิยม ลิสต์ ทูเพิล เซต ได้ แต่มีข้อแม้ว่า Key ต้องไม่ซ้ำกัน นอกจากนี้ Key หรือ Value ยังสามารถสร้างขึ้นมาให้เป็นค่าว่างก่อนได้ แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 5.13
การเขียนคำสั่งโปรแกรมสร้างข้อมูลชนิดดิกชันนามีที่มีค่า Key และ Value เป็นข้อมูลชนิดต่างกัน

```
# Key เป็นข้อมูลชนิดจำนวนเต็ม และ Value เป็นข้อมูลชนิดสตริง
animals = {1: 'Cat', 2: 'Dog', 3: 'Tiger', 4: 'Bird', 5: 'Lion'}

# Key และ Value เป็นข้อมูลชนิดสตริง
sports = {'1': 'Football', '2': 'Tennis', '3': '', '4':
↪  'Basketball'}

# Key ข้อมูลชนิดสตริงและ Value เป็นข้อมูลชนิดทูเพิล
laptops = {'CPU': ('Core i5', 'Core i7', 'M1'), 'Ram': ('8GB',
↪  '16GB', '32GB'), 'SSD': ('512GB', '1TB', '2TB')}

# Key เป็นข้อมูลชนิดทศนิยม และ Value เป็นข้อมูลชนิดสตริง
offices = {1.1: ['เก้าอี้', 'โต๊ะ,'], 1.2: ['ปากกา', 'ดินสอ', 'ยางลบ'],
↪  1.3: ['กรรไกร', 'คัตเตอร์']}

print(animals)  # แสดงผลลัพธ์จากค่าตัวแปร animals
print(sports)   # แสดงผลลัพธ์จากค่าตัวแปร sports
print(laptops)  # แสดงผลลัพธ์จากค่าตัวแปร devices
print(offices)  # แสดงผลลัพธ์จากค่าตัวแปร offices
```

```
{1: 'Cat', 2: 'Dog', 3: 'Tiger', 4: 'Bird', 5: 'Lion'}
{'1': 'Football', '2': 'Tennis', '3': '', '4': 'Basketball'}
{'CPU': ('Core i5', 'Core i7', 'M1'), 'Ram': ('8GB', '16GB',
↪  '32GB'), 'SSD': ('512GB', '1TB', '2TB')}
{1.1: ['เก้าอี้', 'โต๊ะ,'], 1.2: ['ปากกา', 'ดินสอ', 'ยางลบ'], 1.3:
↪  ['กรรไกร', 'คัตเตอร์']}
```

5.3.1 เมธอดและฟังก์ชันที่ใช้กับข้อมูลชนิดดิกชันนารี

ข้อมูลชนิดดิกชันนารีมีเมธอดและฟังก์ชันต่าง ๆ ให้เรียกใช้งานจำนวนมาก เพื่อนำมาใช้ดำเนินการกับข้อมูลที่ถูกจัดเก็บไว้ในตัวแปร หากผู้อ่านต้องการดูรายละเอียดการใช้งานเพิ่มเติม สามารถเรียกดูผ่านคำสั่ง `help(dict)` ได้

เมธอด	ความหมาย	รูปแบบการใช้งาน
`clear()`	ใช้สำหรับลบข้อมูลทั้งหมด	`d.clear()`
`copy()`	ใช้สำหรับคัดลอกข้อมูลชนิดดิกชันนารี	`d.copy()`
`del()`	ใช้สำหรับลบ key หรือตัวแปรดิกชันนารี นับจำนวนข้อมูลที่ซ้ำกันในลิสต์	`del d(key)` หรือ `del d`
`formkeys()`	ใช้สำหรับสร้าง Value ให้เหมือนกันทุกค่า แต่ key จะไม่ซ้ำกัน	`d.fromekeys(key, value=None)` `key` คือ Key ที่ต้องการระบุที่ไม่ซ้ำกัน `value=None` คือ ข้อมูลของแต่ละ Key จะกำหนดหรือไม่มีก็ได้
`get()`	ใช้สำหรับแสดงค่า Value ของ Key ถ้าไม่พบค่า Key ที่ได้ระบุจะคืนค่าเป็น None	`d.get(key, value=None)` `key` คือ Key ที่ต้องการแสดง Value `value=None` คือ Value ของแต่ละ Key จะกำหนดหรือไม่มีก็ได้

ตาราง 5.3: เมธอดและฟังก์ชันสำหรับตัวแปรข้อมูลดิกชันนารี (มีต่อ)

เมธอด	ความหมาย	รูปแบบการใช้งาน
`items()`	ใช้สำหรับแสดงข้อมูลทั้งหมด	`d.items()`
`keys()`	ใช้สำหรับแสดงชื่อ Key ทั้งหมด	`d.keys()`
`pop()`	ใช้สำหรับลบ Value โดยการระบุตำแหน่ง Key โดยจะคืนค่า Value ที่ถูกลบ ถ้าระบุ Key ที่ไม่มีอยู่ในดิกชันนารี จะเกิดการแจ้งเตือนข้อผิดพลาดขึ้น	`d.pop(key [,」 default])` key คือ ค่า Key ที่ต้องการลบค่า Value default คือ ค่าที่ส่งกลับมา ถ้าพบ Key ที่กำหนดจะคืนค่ากลับมาเป็น Value ถ้าไม่พบ Key ที่กำหนด จะคืนค่ากลับมาเป็นค่าที่กำหนดในส่วน `default` ถ้าไม่ได้กำหนดจะคืนค่ากลับมาเป็นการแจ้งเตือนข้อผิดพลาด
`popitem()`	ใช้สำหรับลบข้อมูลตัวสุดท้ายออกจากดิกชันนารี จะคืนค่าคู่ Key และ Value กลับมา ถ้าลบค่าข้อมูลในดิกชันนารีที่เก็บค่าว่างจะเกิดการแจ้งเตือนข้อผิดพลาด	`d.popitem()`

ตาราง 5.3: เมธอดและฟังก์ชันสำหรับตัวแปรข้อมูลดิกชันนารี (มีต่อ)

เมธอด	ความหมาย	รูปแบบการใช้งาน
`setdefault()`	ใช้สำหรับค้นหาและเพิ่มข้อมูลให้กับดิกชันนารี แต่มีข้อแม้ว่า Key ต้องไม่ซ้ำกับ Key ที่มีอยู่ ถ้าพบว่ามี Key อยู่แล้วจะไม่ทำการเพิ่มข้อมูลหรือเปลี่ยนแปลง Value แต่ถ้าไม่พบ Key ถึงจะเพิ่มค่าข้อมูลใหม่ และหากไม่กหนด Value จะกำหนดเป็นค่า None	`d.setdefault(key, [default=None])` `key` คือ Key ที่ต้องการสร้างและเพิ่ม Value `default=None` คือ Value ที่ต้องการเพิ่ม หากไม่กำหนดจะเป็นค่า None
`update()`	ใช้สำหรับแก้ไขข้อมูลในดิกชันนารี และจะทำการเขียนทับหากมี Key นั้นอยู่แล้ว	`d.update([other])` `other` คือ ตัวแปรชนิดดิกชันนารี เช่น `d.update({'cat': 2})` หรือทูเพิลที่มีความยาวเป็น 2 เช่น `d.update('cat', 2)` หรือการให้ค่าในรูปแบบ key/value เช่น `d.update(cat=2)`
`values()`	ใช้สำหรับแสดงค่า Value ทั้งหมดในดิกชันนารี	`d.values()`
`len()`	เป็นฟังก์ชันที่ใช้แสดงจำนวนสมาชิกในดิกชันนารี	`len(d)`

ตาราง 5.3: เมธอดและฟังก์ชันสำหรับตัวแปรข้อมูลดิกชันนารี

ตัวอย่าง 5.14
การเขียนคำสั่งโปรแกรมการใช้เมธอดและฟังก์ชันกับข้อมูลชนิดดิกชันนารี

```
# Key เป็นข้อมูลชนิดจำนวนเต็ม และ Value เป็นข้อมูลชนิดสตริง
animals = {1: 'Cat', 2: 'Dog', 3: 'Rat', 4: 'Bird', 5: 'Lion'}
# Key และ Value เป็นข้อมูลชนิดสตริง
sports = {'1': 'Football','2': '', '3': 'Golf','4':
↪  'Basketball'}
# ใช้เมธอด get() แสดงค่าข้อมูล Value ของ animals
print ('ตำแหน่งที่ 2 ของ animals คือ', animals.get(2))
# ใช้เมธอด key() แสดงค่า key ทั้งหมดในค่าตัวแปร sports
print ('key ที่มีอยู่ใน soprts คือ', sports.keys())
# แสดงจำนวนข้อมูลทั้งหมดด้วยฟังก์ชัน len() ของ animals
print ('ข้อมูลทั้งหมดใน animals คือ', len(animals))
# แสดงค่า Value ทั้งหมดด้วยเมธอด values() ใน sports
print ('ค่า value ที่อยู่ใน sports = ', sports.values())
```

```
ตำแหน่งที่ 2 ของ animals คือ Dog
key ที่มีอยู่ใน soprts คือ dict_keys(['1', '2', '3', '4'])
ข้อมูลทั้งหมดใน animals คือ 5
ค่า value ที่อยู่ใน sports =  dict_values(['Football', '', 'Golf',
↪  'Basketball'])
```

5.3.2 การดำเนินการกับข้อมูลชนิดดิกชันนารี

นอกจากเมธอดต่าง ๆ ที่ได้กล่าวมาแล้วนั้น เรายังดำเนินการกับข้อมูลชนิดดิกชันนารีได้อีก หากเราต้องการลบข้อมูลออกจากตัวแปรทั้งค่า Key และค่า Value ให้ใช้คำสั่ง del แต่จะมีการแจ้งเตือนเกิดขึ้นเมื่อระบุตำแหน่งหรือระบุตัวแปรที่ต้องการลบไม่ถูกต้อง มีรูปแบบการใช้งานดังนี้

โครงสร้างคำสั่งการลบข้อมูลออกจากตัวแปรชนิดดิกชันนารี

```
del dict.[key]
```

`dict`	ตัวแปรข้อมูลชนิดดิกชันนารีที่ต้องลบ ออบเจ็คที่ต้องการแสดงผล มีได้มากกว่า 1 ออบเจ็ค
`key`	Key ที่ต้องการลบ หากไม่ระบุ จะเป็นการลบค่าตัวแปรนั้นทั้งหมด

ตัวอย่าง 5.15
การเขียนคำสั่งโปรแกรมลบดิกชันนารี โดยการคำสั่ง del

```
1 sports = {'1':'Football', '2':'Tennis', '3':'Basketball',
2 '4':'Golf'}
3 del sports['4']  # ใช้คำสั่ง del ลบข้อมูล Key '4' ของ sports
4 print('ข้อมูลที่อยู่ใน sports =', sports)
5 del sports  # ใช้คำสั่ง del ตัวแปร sports
6 print(sports)
```

```
ข้อมูลที่อยู่ใน sports = {'1': 'Football', '2': 'Tennis', '3':
↪  'Basketball'}
Traceback (most recent call last):
  File "src/test.py", line 6, in <module>
    print(sports)
NameError: name 'sports' is not defined
```

จากตัวอย่าง 5.15 จะเห็นว่าหลังจากเราทำการลบข้อมูล `sports` ในตำแหน่งที่ key `'4'` ในบรรทัดที่ 3 ทำให้ `sports` มีสมาชิกเหลือเหลือ 3 ชุด และในบรรทัดที่ 5 เราทำการลบข้อมูลตัวแปรดิกชันนารี `sports` ทั้งหมด ทำให้โปรแกรมแสดงข้อผิดพลาดว่าไม่มีตัวแปรชื่อ `sports` เมื่อเราต้องการแสดงค่า `sports` ในบรรทัดที่ 6

เมื่อเราต้องการแก้ไขข้อมูลในส่วนของ Value หรือเพิ่มข้อมูลเข้าไปเก็บไว้ในตัวแปรดิกชันนารี เราสามารถใช้คำสั่ง `dict[key]=value` โดยที่ `key` คือ Key ที่ต้องการแก้ไข หรือเพิ่มข้อมูลใหม่ ถ้า Key มีอยู่แล้วจะเป็นการเพิ่มข้อมูลใหม่ ส่วน `value` คือ ค่าข้อมูลที่ต้องการแก้ไข

ตัวอย่าง 5.16
การเขียนคำสั่งโปรแกรมเปลี่ยนข้อมูลของตัวแปรดิกชันนารี

```
# Key และ Value เป็นข้อมูลชนิดสตริง
sports = {'1':'Football', '2':'Tennis','3':'Basketball'}
# แก้ไข Value ที่ Key 1 จาก Football เป็น Racing
sports['1'] = 'Golf'
# เพิ่มสมาชิกเข้าไปในค่าตัวแปร sports
sports['4'] = 'Rugby'
# แสดงผลลัพธ์จากการแก้ไขและเพิ่มข้อมูล
print('รายชื่อชนิดกีฬาที่มีอยู่ใน sports =', sports)
```

```
รายชื่อชนิดกีฬาที่มีอยู่ใน sports = {'1': 'Golf', '2': 'Tennis', '3':
↪  'Basketball', '4': 'Rugby'}
```

5.4 รู้จักกับข้อมูลชนิดเซต

การสร้างข้อมูลชนิดเซต (Set) ขึ้นมาใช้งาน จะเป็นการจัดเก็บกลุ่มข้อมูลประเภทเดียวกันหรือคล้ายกันให้อยู่ด้วยกัน เช่น

```
กลุ่มข้อมูลสัตว์ปีก = {'ไก่', 'เป็ด', 'นก', 'ห่าน'}
กลุ่มข้อมูลยี่ห้อรถยนต์ = {'Toyota', 'Honda', 'BMW', 'Porsche'}
กลุ่มข้อมูลกีฬา = {'Football', 'Basketball', 'Golf', 'Tennis'}
```

เป็นต้น ข้อมูลที่ถูกเก็บในข้อมูลชนิดเซตจะอยู่ในเครื่องหมายปีกกา `{...}` และถูกคั่นด้วยเครื่องหมาย Comma (`,`) ข้อมูลชนิดเซตจัดเก็บข้อมูลแบบไม่มีลำดับ (Unordered Collection) แบ่งออกเป็น 2 ประเภทได้แก่ Set และ Frozenset แต่มีความแตกต่างกันคือ

Set สามารถเปลี่ยนแปลงแก้ไขสมาชิกได้ แต่สำหรับ Frozenset นั้น ไม่สามารถแก้ไขสมาชิกในเซตได้ กรณีที่สร้างเซตขึ้นมาใช้งานและมีสมาชิกซ้ำกัน และเมื่อสั่งประมวลผลสมาชิกที่ซ้ำกันจะเหลือเพียงแค่ตัวเดียวเท่านั้น

ตัวอย่าง 5.17
การเขียนคำสั่งโปรแกรมสร้างข้อมูลชนิดเซตและการตรวจสอบข้อมูลชนิดเซต

```
# สร้างตัวแปร sport เป็นข้อมูลชนิดเซตที่เป็นเซตว่าง
sport = set()
# สร้างตัวแปรเป็นข้อมูลชนิดเซตของข้อมูลชนิดสตริง
books = {'Python', 'C', 'C++', 'Java', 'C++'}
# สร้างตัวแปรเป็นข้อมูลชนิดเซตเป็นข้อมูลชนิดจำนวนเต็ม
numbers = {2, 5, 11, 7, 3, 13, 17, 19}
# แสดงผลลัพธ์ค่าตัวแปร books
print('สมาชิกของ books ประกอบด้วย', books)
# แสดงผลลัพธ์ค่าตัวแปร numbers
print('สมาชิกของ numbers ประกอบด้วย', numbers)
# ตรวจสอบข้อมูลชนิดด้วยฟังก์ชัน type()
print('ข้อมูลชนิดของ sport คือ', type(sport))
# ตรวจสอบข้อมูลชนิดด้วยฟังก์ชัน type()
print('ข้อมูลชนิดของ books คือ', type(books))
```

```
สมาชิกของ books ประกอบด้วย {'C', 'C++', 'Java', 'Python'}
สมาชิกของ numbers ประกอบด้วย {2, 3, 5, 7, 11, 13, 17, 19}
ข้อมูลชนิดของ sport คือ <class 'set'>
ข้อมูลชนิดของ books คือ <class 'set'>
```

จากตัวอย่าง 5.17 ในบรรทัดที่ 4 ตัวแปร books มี 'C++' ปรากฏอยู่ 2 ครั้ง เมื่อแสดงผลออกมาจะทำให้ข้อมูลที่ซ้ำกันเหลือเพียงค่าเดียว ในบรรทัดที่ 6 เราสร้างตัวแปร numbers เมื่อทำการแสดงผลผ่านคำสั่ง print() ผลลัพธ์ที่ได้ในแต่ละครั้งอาจจะมีการเรียงลำดับต่างออกไป ซึ่งเป็นการสุ่มสมาชิกของเซตมาแสดงผลแบบไม่มีลำดับ

ตัวอย่าง 5.18
ข้อมูลชนิดเซตเป็นข้อมูลแบบไม่มีลำดับ

```
1 # สร้างตัวแปรเป็นข้อมูลชนิดเซตของข้อมูลชนิดสตริง
2 books = {'Python', 'C', 'C++', 'Java', 'C++'}
3 # แสดงข้อมูลตัวแปร books
4 print('ตัวแปร books คือ', books)
5 # การเข้าถึงข้อมูลตำแหน่งแรกของ books
6 print('ข้อมูลตำแหน่งแรกของ books คือ', books[0])
```

```
ตัวแปร books คือ {'Java', 'C', 'C++', 'Python'}
Traceback (most recent call last):
  File "src/test.py", line 6, in <module>
    print('ข้อมูลตำแหน่งแรกของ books คือ', books[0])
TypeError: 'set' object is not subscriptable
```

จากตัวอย่าง 5.18 ในบรรทัดที่ 4 เมื่อพยายามเข้าถึงข้อมูลตำแหน่งแรกของตัวแปร `books` โปรแกรมจะแสดงข้อผิดพลาดขึ้น เนื่องจากข้อมูลชนิดเซตจัดเก็บข้อมูลแบบไม่มีลำดับ ภาษาโปรแกรมไพธอนไม่อนุญาตให้ใช้เครื่องหมาย `[...]` ในการเข้าถึงได้

จากที่ได้นำเสนอตัวอย่างข้างต้นเป็นการสร้างข้อมูลชนิดเซต ซึ่งเป็นข้อมูลชนิดที่สามารถแก้ไขหรือลบสมาชิกออกจากเซตได้ และมีอีกหนึ่งข้อมูลชนิดคือ Frozenset ซึ่งเป็นข้อมูลชนิดที่มีการเก็บสมาชิกเหมือนกับข้อมูลชนิดเซต แต่สมาชิกภายในเซตจะไม่สามารถแก้ไขเปลี่ยนแปลงได้ หากพยายามแก้ไขสมาชิกจะเกิดการแจ้งเตือนข้อผิดพลาดขึ้น การสร้างข้อมูลชนิด Frozenset ขึ้นมาใช้งานจะอยู่ในรูปแบบของ `frozensent()` แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 5.19
การเขียนคำสั่งโปรแกรมสร้างข้อมูลชนิด Frozenset

```
# สร้างตัวแปร cities เป็นข้อมูลชนิด frozenset
cities = frozenset(('Bangkok', 'Berlin', 'Tokyo', 'New York'))
# สร้างตัวแปร colors เป็นข้อมูลชนิด frozenset
colors = frozenset(['Red', 'Green', 'Blue'])
# แสดง types และ สมาชิกที่เก็บอยู่ในค่าตัวแปร cities
print('cities = ', cities, 'which type =', type(cities))
# แสดง types และ สมาชิกที่เก็บอยู่ในค่าตัวแปร colors
print('colors = ', colors, 'which type =', type(colors))
```

```
cities =  frozenset({'New York', 'Bangkok', 'Tokyo', 'Berlin'})
↪  which type = <class 'frozenset'>
colors =  frozenset({'Green', 'Blue', 'Red'}) which type =
↪  <class 'frozenset'>
```

5.4.1 เมธอดและฟังก์ชันที่ใช้กับข้อมูลชนิดเซต

ข้อมูลชนิดเซตมีเมธอดต่าง ๆ ที่นำมาใช้จัดการกับสมาชิกที่อยู่ภายในเซตจำนวนมาก ผู้อ่านสามารถใช้คำสั่ง `help(set)` ขอดูข้อมูลการใช้งานเมธอดกับข้อมูลชนิดนี้เพิ่มเติมได้

เมธอด	ความหมาย	รูปแบบการใช้งาน
`add()`	เพิ่มสมาชิกเข้าไปในเซต	`s.add()`
`clear()`	ลบสมาชิกทั้งหมดในเซต	`s.clear()`
`copy()`	คัดลอกข้อมูลชนิดเซต	`s.copy()`
`difference()`	หาสมาชิกที่ต่างกันระหว่างสองเซต หรือใช้ตัวดำเนินการ `-` แทนได้	`s1.difference(s2)`

ตาราง 5.4: เมธอดและฟังก์ชันของข้อมูลชนิดเซต (มีต่อ)

เมธอด	ความหมาย	รูปแบบการใช้งาน
difference_update()	หาสมาชิกที่ต่างกันระหว่างสองเซต แต่เซตที่นำมาเปรียบเทียบจะเก็บค่าสมาชิกใหม่	s1.difference_update(s2)
discard()	ลบสมาชิกในเซต	s.discard(x) x คือ ชื่อสมาชิกที่ต้องการลบ
intersection()	หาสมาชิกที่เหมือนกันระหว่างเซต สามารถใช้เครื่องหมาย & แทนได้	s1.intersection(s2, ..., sn)
intersection_update()	หาสมาชิกที่เหมือนกันระหว่างสองเซต ผลลัพธ์ที่ได้จะถูกนำไปแทนที่สมาชิกของเซตที่นำมาหาสมาชิกที่เหมือนกัน	s1.intersection_update(s2)
isdisjoint()	ตรวจสอบสมาชิกภายในเซตว่าเป็นสมาชิกภายในเซตว่าเป็นสมาชิกของเซตอื่นด้วยหรือไม่ ถ้าเป็นสมาชิกของเซตอื่นผลลัพธ์เป็น False ถ้าไม่เป็นสมาชิกของเซตอื่นผลลัพธ์เป็น True	s1.isdisjoint(s2)

ตาราง 5.4: เมธอดและฟังก์ชันของข้อมูลชนิดเซต (มีต่อ)

เมธอด	ความหมาย	รูปแบบการใช้งาน
`issubset()`	ตรวจสอบการเป็นสับเซต (subset) หรือเซตย่อย ถ้าสมาชิกทุกตัวของ `s1` เป็นสมาชิกของ `s2` จะได้ผลลัพธ์เป็น `True` ถ้ามีบางตัวไม่เป็นสมาชิกของตัวแปร `s2` จะได้ผลลัพธ์เป็น `False`	`s1.issubset(s2)`
`issuperset()`	ตรวจสอบการเป็นซุปเปอร์เซตของเซต ถ้าเป็นซุปเปอร์เซตให้ผลลัพธ์เป็น `True` ถ้าไม่เป็นซุปเปอร์เซตให้ผลลัพธ์เป็น `False`	`s1.issuperset(s2)`
`pop()`	ลบสมาชิกออกจากเซตโดยการสุ่ม จะคืนค่ากลับมาเป็นสมาชิกที่ถูกลบ ถ้าลบสมาชิกในเซตว่างจะคืนค่ากลับมาเป็นการแจ้งเตือนข้อผิดพลาด	`s.pop()`
`remove()`	ลบสมาชิกออกจากเซตเหมือนกับเมธอด `discard()` แต่เมื่อนำเมธอดนี้มาใช้งาน ถ้าลบสมาชิกที่ไม่มีอยู่ในเซตจะเกิดการแจ้งเตือนข้อผิดพลาด	`s.remove(x)` `x` คือ ชื่อสมาชิกที่ต้องการลบ
`symmetric_difference()`	หาสมาชิกที่ไม่มีอยู่ในอีกเซต สามารถใช้เครื่องหมาย `(^)` แทนได้	`s1.symmetric_difference(s2)`

ตาราง 5.4: เมธอดและฟังก์ชันของข้อมูลชนิดเซต (มีต่อ)

เมธอด	ความหมาย	รูปแบบการใช้งาน
`symmetric_difference_update()`	รวมสมาชิกของเซตเข้าด้วยกัน สามารถใช้เครื่องหมาย `|` แทนได้ ถ้ามีสมาชิกที่เหมือนกันจะเหลือเพียงตัวเดียว	`s1.symmetric_difference_update(s2)`
`union()`	update สมาชิกของเซต หรือเป็นการรวมสมาชิกสองเซต ถ้าสมาชิกมีซ้ำกันจะถูกตัดออกให้เหลือเพียงตัวเดียว	`s1.union(s2)`
`update()`	update สมาชิกของเซต หรือเป็นการรวมสมาชิกสองเซต ถ้าสมาชิกมีซ้ำกันจะถูกตัดออกให้เหลือเพียงตัวเดียว	`s1.update(s2)`
`len()`	ฟังก์ชันแสดงจำนวนสมาชิกทั้งหมดในเซต	`len(s)`

ตาราง 5.4: เมธอดและฟังก์ชันของข้อมูลชนิดเซต

5.4.2 การดำเนินการกับข้อมูลชนิดเซต

นอกจากเราสามารถนำเมธอดข้างต้นมาใช้กับข้อมูลชนิดเซตแล้ว เรายังสามารถนำฟังก์ชัน `min()`, `max()` และ `sum()` ตัวดำเนินการเปรียบเทียบ ตัวดำเนินการเป็นสมาชิก หรือตัวดำเนินการเอกลักษณ์ เข้ามาดำเนินการกับข้อมูลชนิดเซตได้อีกด้วย แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 5.20
การใช้ฟังก์ชัน `min()`,`max()` และ `sum()` กับข้อมูลชนิดเซต

```
# สร้างตัวแปรข้อมูลชนิดเซตเป็นจำนวนเต็ม
nums = {2, 3, 5, 7, 11}
# แสดงผลลัพธ์ค่าที่น้อยที่สุดด้วยฟังก์ชัน min()
print ('จำนวนที่น้อยที่สุดใน nums คือ', min(nums))
# แสดงผลลัพธ์ค่าที่มากที่สุดด้วยฟังก์ชัน max()
print ('จำนวนที่มากที่สุดใน nums คือ', max(nums))
# แสดงผลลัพธ์ค่าผลรวมด้วยฟังก์ชัน sum()
print ('ผลรวมสมาชิกใน nums คือ', sum(nums))
```

```
จำนวนที่น้อยที่สุดใน nums คือ 2
จำนวนที่มากที่สุดใน nums คือ 11
ผลรวมสมาชิกใน nums คือ 28
```

สำหรับตัวดำเนินการเปรียบเทียบ `<=` และ `<` สำหรับข้อมูลชนิดเซตจะหมายถึงการตรวจสอบว่าเป็น ซับเซต (Subset) และซับเซตแท้ (Proper Subset) หรือไม่ตามลำดับ ซึ่งความหมายของซับเซตและซับเซตแท้นี้จะนิยามตามความหมายทางคณิตศาสตร์ ในทำนองเดียวกันกับตัวดำเนินการเปรียบเทียบ `>=` และ `>` สำหรับข้อมูลชนิดเซตจะหมายถึงการตรวจสอบว่าเป็น ซุปเปอร์เซต (Superset) และซุปเปอร์เซตแท้ (Proper Superset) หรือไม่ตามลำดับ และสำหรับตัวดำเนินการเปรียบเทียบ `==` สำหรับข้อมูลชนิดเซตจะหมายถึงการตรวจสอบว่าตัวแปรเซตทั้งสองตัวนั้นมีข้อมูลเหมือนกันใช่หรือไม่

ตัวอย่าง 5.21
การใช้ตัวดำเนินการเปรียบเทียบกับข้อมูลชนิดเซต

```
A = {2, 3, 5, 7, 11}
B = {3, 2, 7, 5}
C = {7, 5, 3, 2}

print('A เป็นซับเซตแท้ของ B หรือไม่', A < B)
print('A เป็นซับเซตของ B หรือไม่', A <= B)
print('A เป็นซุปเปอร์เซตแท้ของ B หรือไม่', A > B)
print('A เป็นซุปเปอร์เซตของ B หรือไม่', A >= B)
print('B เป็นซับเซตแท้ของ C หรือไม่', B < C)
print('B เป็นซับเซตของ C หรือไม่', B <= C)
print('4 เป็นสมาชิกของ A หรือไม่', 4 in A)
print('0 ไม่เป็นสมาชิกของ A หรือไม่', 0 not in A)
print('A เท่ากับ B หรือไม่', A == B)
print('B เท่ากับ C หรือไม่', B == C)
print('B คือออบเจ็คเดียวกับ C หรือไม่', B is C)
```

```
A เป็นซับเซตแท้ของ B หรือไม่ False
A เป็นซับเซตของ B หรือไม่ False
A เป็นซุปเปอร์เซตแท้ของ B หรือไม่ True
A เป็นซุปเปอร์เซตของ B หรือไม่ True
B เป็นซับเซตแท้ของ C หรือไม่ False
B เป็นซับเซตของ C หรือไม่ True
4 เป็นสมาชิกของ A หรือไม่ False
0 ไม่เป็นสมาชิกของ A หรือไม่ True
A เท่ากับ B หรือไม่ False
B เท่ากับ C หรือไม่ True
B คือออบเจ็คเดียวกับ C หรือไม่ False
```

5.5 การแปลงข้อมูลชนิดลิสต์ ทูเพิล ดิกชันนารี และเซต

ในบางครั้งเราอาจจำเป็นต้องทำการแปลง (Casting) ข้อมูลชนิดหนึ่งไปเป็นอีกชนิดหนึ่ง เช่น เมื่อเราต้องการกำจัดข้อมูลที่ซ้ำกันในข้อมูลชนิดลิสต์ เราอาจจะแปลงข้อมูลนั้นไปเป็นข้อมูลชนิดเซตก่อน แล้วค่อยแปลงกลับไปเป็นข้อมูลชนิดลิสต์ดังเดิม ซึ่งเราสามารถใช้ฟังก์ชันต่อไปนี้ในการแปลงข้อมูลชิดต่าง ๆ ได้

- ฟังก์ชัน `list()` ใช้ในการแปลงข้อมูลเป็นข้อมูลชนิดลิสต์
- ฟังก์ชัน `tuple()` ใช้ในการแปลงข้อมูลเป็นข้อมูลชนิดทูเพิล
- ฟังก์ชัน `dict()` ใช้ในการแปลงข้อมูลเป็นข้อมูลชนิดดิกชันนารี
- ฟังก์ชัน `set()` และ `frozenset()` ใช้ในการแปลงข้อมูลเป็นข้อมูลชนิดเซต

ตัวอย่าง 5.22
การเขียนคำสั่งโปรแกรมแปลงข้อมูลเป็นข้อมูลชนิดลิสต์

```
fnums = [3.50, 17.475, 15.899]
text = 'Python'
snums = {2, 3, 5, 7}
sports= {'1': 'Football', '2': 'Tennis', '3': 'Basketball'}
print('แปลงลิสต์เป็นลิสต์ ->', list(fnums))
print('แปลงสตริงเป็นลิสต์ ->', list(text))
print('แปลงเซตเป็นลิสต์ ->', list(snums))
print('แปลง Key ในดิกชันนารีเป็นลิสต์  ->', list(sports.keys()))
print('แปลง Value ในดิกชันนารีเป็นลิสต์ ->', list(sports.values()))
```

```
แปลงลิสต์เป็นลิสต์ -> [3.5, 17.475, 15.899]
แปลงสตริงเป็นลิสต์ -> ['P', 'y', 't', 'h', 'o', 'n']
แปลงเซตเป็นลิสต์ -> [2, 3, 5, 7]
แปลง Key ในดิกชันนารีเป็นลิสต์  -> ['1', '2', '3']
แปลง Value ในดิกชันนารีเป็นลิสต์ -> ['Football', 'Tennis', 'Basketball']
```

ตัวอย่าง 5.23
การเขียนคำสั่งโปรแกรมแปลงข้อมูลเป็นข้อมูลชนิดทูเพิล

```
fnums = [3.50, 17.475, 15.899]
text = 'Python'
snums = {2, 3, 5, 7}
sports= {'1': 'Football', '2': 'Tennis', '3': 'Basketball'}
print('แปลงลิสต์เป็นทูเพิล ->', tuple(fnums))
print('แปลงสตริงเป็นทูเพิล ->', tuple(text))
print('แปลงเซตเป็นทูเพิล ->', tuple(snums))
print('แปลง Key ในดิกชันนารีเป็นทูเพิล  ->', tuple(sports.keys()))
print('แปลง Value ในดิกชันนารีเป็นทูเพิล ->', tuple(sports.values()))
```

```
แปลงลิสต์เป็นทูเพิล -> (3.5, 17.475, 15.899)
แปลงสตริงเป็นทูเพิล -> ('P', 'y', 't', 'h', 'o', 'n')
แปลงเซตเป็นทูเพิล -> (2, 3, 5, 7)
แปลง Key ในดิกชันนารีเป็นทูเพิล  -> ('1', '2', '3')
แปลง Value ในดิกชันนารีเป็นทูเพิล -> ('Football', 'Tennis', 'Basketball')
```

ตัวอย่าง 5.24
การเขียนคำสั่งโปรแกรมแปลงข้อมูลเป็นข้อมูลชนิดดิกชันนารี

```
animals = [[1, 'Cat'], [2, 'Dog'], [3, 'Bee'], [4, 'Ant']]
sports = ((1, 'Football'), (2, 'Basketball'))
numbers = {2, 3, 5, 7}
print('แปลงลิสต์เป็นดิกชันนารี ->', dict(animals))
print('แปลงทูเพิลเป็นดิกชันนารี ->', dict(sports))
print('แปลงเซตเป็นดิกชันนารี ->', dict.fromkeys(numbers))
```

```
แปลงลิสต์เป็นดิกชันนารี -> {1: 'Cat', 2: 'Dog', 3: 'Bee', 4: 'Ant'}
แปลงทูเพิลเป็นดิกชันนารี -> {1: 'Football', 2: 'Basketball'}
แปลงเซตเป็นดิกชันนารี -> {2: None, 3: None, 5: None, 7: None}
```

ตัวอย่าง 5.25
การเขียนคำสั่งโปรแกรมแปลงข้อมูลเป็นข้อมูลชนิดเซต

```
fnums = (3.14, 16.0, 3.14)
books = ['Python', 'Java', 'C', 'C++']
text = 'Python'
numbers = '112'
print('แปลงทูเพิลเป็นเซต ->', set(fnums))
print('แปลงลิสต์เป็นเซต ->', set(books))
print('แปลงสตริงเป็นเซต ->', set(text))
print('แปลงสตริงเป็นเซต ->', set(numbers))
print('แปลงทูเพิลเป็นเซต ->', frozenset(fnums))
print('แปลงลิสต์เป็นเซต ->', frozenset(books))
print('แปลงสตริงเป็นเซต ->', frozenset(text))
print('แปลงสตริงเป็นเซต ->', frozenset(numbers))
```

```
แปลงทูเพิลเป็นเซต -> {16.0, 3.14}
แปลงลิสต์เป็นเซต -> {'C', 'C++', 'Python', 'Java'}
แปลงสตริงเป็นเซต -> {'n', 't', 'o', 'P', 'y', 'h'}
แปลงสตริงเป็นเซต -> {'1', '2'}
แปลงทูเพิลเป็นเซต -> frozenset({16.0, 3.14})
แปลงลิสต์เป็นเซต -> frozenset({'C', 'C++', 'Python', 'Java'})
แปลงสตริงเป็นเซต -> frozenset({'n', 't', 'o', 'P', 'y', 'h'})
แปลงสตริงเป็นเซต -> frozenset({'1', '2'})
```

สรุปก่อนจบบท

เนื้อหาในบทนี้ผู้อ่านได้รู้จักกับโครงสร้างข้อมูลภาษาไพธอน ได้แก่ ข้อมูลชนิดลิสต์ ข้อมูลชนิดทูเพิล ข้อมูลชนิดดิกชันนารี และข้อมูลชนิดเซต ซึ่งเป็นข้อมูลชนิดที่มีการจัดเก็บข้อมูลแบบลำดับและสามารถจัดเก็บข้อมูลได้หลายชนิด อย่างไรก็ตามข้อมูลชนิดเหล่านี้มีความแตกต่างกันคือการแก้ไขหรือเปลี่ยนแปลงข้อมูล ดังนั้นผู้อ่านจะต้องระมัดระวังหรือเลือกข้อมูลชนิดเพื่อใช้งานให้ถูกต้อง

แบบฝึกหัด

1. กำหนดให้

```
mounts=['Jan', 'May', 'Jul', 'Aug', 'Oct', 'Dec']
```

   (a) ให้เขียนคำสั่งโปรแกรมเรียกใช้เมธอดแทรกเดือนที่ขาดหายไป

   (b) ให้เขียนคำสั่งโปรแกรมเรียกใช้เมธอดลบชื่อเดือนตำแหน่งที่ 2, 5, 9 ออกจากลิสต์

   (c) ให้เขียนคำสั่งโปรแกรมเรียกใช้เมธอดแสดงชื่อเดือนที่เหลืออยู่ในลิสต์

2. กำหนดให้

```
days = ['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri' , 'Sat']
```

   (a) ให้เขียนคำสั่งโปรแกรมเรียกใช้เมธอดเรียงชื่อวันจากท้ายสุด

   (b) ให้เขียนคำสั่งโปรแกรมเรียกใช้เมธอดเรียงลำดับชื่อวันตามตัวอักษร

   (c) ให้เขียนคำสั่งโปรแกรมแสดงชื่อวันในตำแหน่งที่ 0, 5 และ 6

3. กำหนดให้

```
brand_cars = ('Toyota', 'Honda', 'Benz', 'BMW',
↪  'Tesla', 'Ford' , 'KIA', 'Volvo')
```

   (a) ให้เขียนคำสั่งโปรแกรมแสดงตำแหน่งของ `'Benz'`, `'Ford'` และ `'Volvo'`

   (b) ให้เขียนคำสั่งโปรแกรมแสดงจำนวนข้อมูลทั้งหมดในทูเพิล

   (c) ให้เขียนคำสั่งโปรแกรมตรวจสอบมียี่ห้อรถ `'Suzuki'`, `'Ferrary'`, `'Ford'` อยู่ใน `brand_cars` หรือไม่

4. จงเขียนคำสั่งโปรแกรมสร้างข้อมูลชนิดดิกชันนารี เก็บรายชื่อเดือนทั้งหมดพร้อมทั้งแสดงตัวอย่างการเรียกใช้งานเมธอดดังนี้

   (a) เรียกใช้งานเมธอดแสดงค่าข้อมูลของ Key ในเดือนมกราคม, ตุลาคม และธันวาคม

(b) เรียกใช้งานเมธอดลบค่า Key ในเดือนมีนาคม และ พฤศจิกายน

(c) เรียกใช้งานเมธอดแสดงค่า Value ทั้งหมดในดิกชันนารี

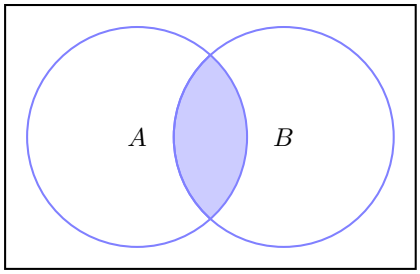
5. จงเขียนคำสั่งโปรแกรมสร้างข้อมูลชนิดดิกชันนารีเก็บรายชื่อเพื่อนที่สนิทจำนวน 3 คน กำหนดให้ค่าข้อมูล Value ต้องประกอบด้วยชื่อ – นามสกุล ชื่อเล่น และเบอร์โทรศัพท์ และให้แสดงตัวอย่างการเรียกใช้งานเมธอดดังนี้

(a) เรียกใช้งานเมธอดเพิ่มชื่อเพื่อนเข้าไปในดิกชันนารีอีก 2 คน และแสดงผลข้อมูลทั้งหมดที่มีอยู่ในดิกชันนารี

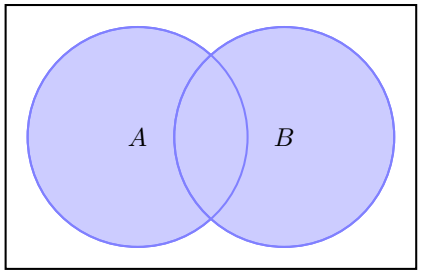
(b) ให้เขียนคำสั่งโปรแกรมแสดงชื่อเพื่อนทั้งหมดที่มีอยู่ในดิกชันนารี

(c) ให้เขียนคำสั่งโปรแกรมแสดงชื่อเพื่อนและเบอร์โทรศัพท์ที่มีอยู่ในดิกชันนารี

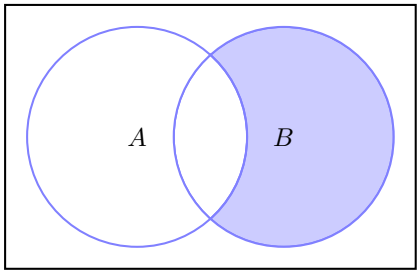
6. กำหนดให้ A และ B เป็นตัวแปรของข้อมูลชนิดเซต จงเขียนนิพจน์ในภาษาโปรแกรมไพธอนที่สอดคล้องกับแผนภาพเวนน์-ออยเลอร์ที่กำหนดให้ต่อไปนี้



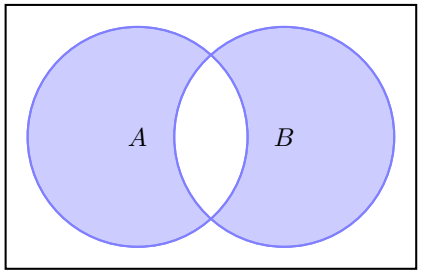
(a)



(b)



(c)



(d)

7. กำหนดให้

```
x = {'cat', 'dog', 'fish', 'bird', 'bee'}
```

```
y = {'snake', 'lion', 'pig', 'dog', 'cat'}
```

   (a) จงเขียนคำสั่งโปรแกรมหาผลลัพธ์สมาชิกที่เหมือนกันของเซต x และ y

   (b) จงเขียนคำสั่งโปรแกรมหาสมาชิกที่อยู่ในเซต x แต่ไม่อยู่ในเซต y

   (c) จงเขียนคำสั่งโปรแกรมหาสมาชิกที่อยู่ในเซต x แต่ไม่อยู่ในเซต y และอยู่ในเซต y แต่ไม่อยู่ในเซต x