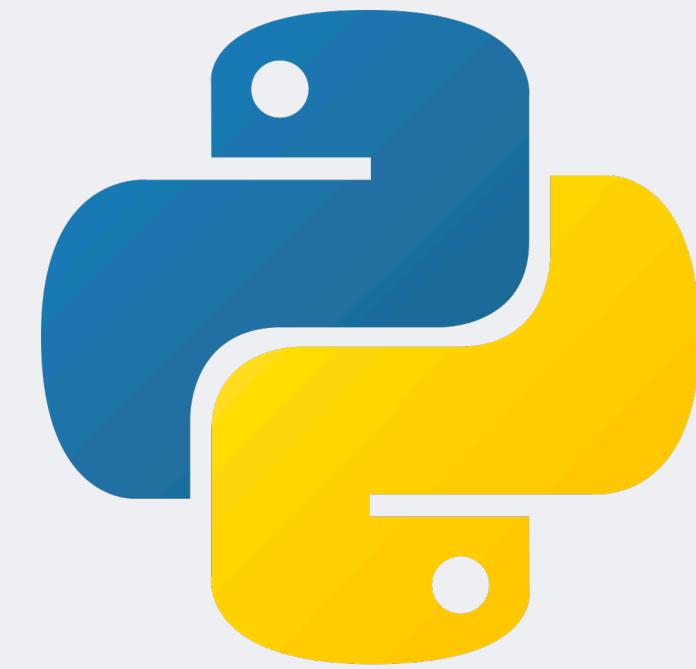


STR EXTRA

```
def sort_words_in_sentence(sentence):
    words = sentence.split()                      # Split sentence into words
    words.sort(key=str.lower)                     # Sort words alphabetically (case-insensitive)
    sorted_sentence = ' '.join(words)            # Join them back into a sentence
    return sorted_sentence

# Example usage
sentence = "This is a man world"
sorted_result = sort_words_in_sentence(sentence)
print("Sorted sentence:", sorted_result)
```

Sorted sentence: a is man This world



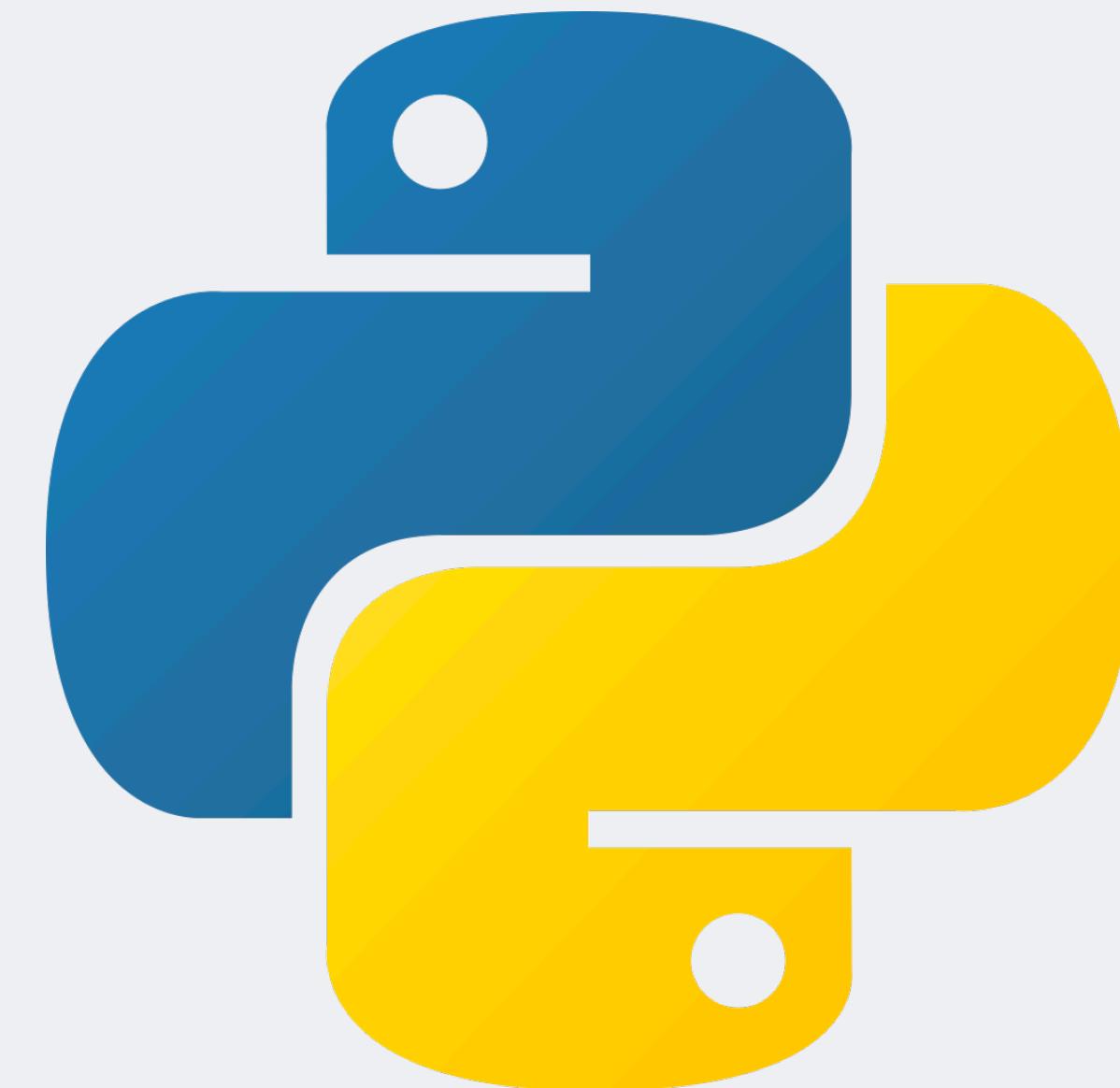
INE-PYTHON

LECTURE - 07 SET AND DICTIONARY

What You Will Learn

- Sets
- Dictionaries

SET



Set Methods and Useful Built-in Functions

SET

เซต (Set): วัตถุที่ใช้เก็บกลุ่มของข้อมูลในลักษณะเดียวกับเซตทางคณิตศาสตร์

- รายการทั้งหมดต้องไม่ซ้ำกัน
- เซตไม่มีลำดับ
- สมาชิกในเซตสามารถเป็นข้อมูลคนละประเภทกันได้

SET

แนวคิดสำคัญของเซต:

- กลุ่มข้อมูลที่ไม่เรียงลำดับ: เซตไม่รักษาลำดับของข้อมูลที่เก็บอยู่
- ข้อมูลไม่ซ้ำกัน: เซตจะลบค่าที่ซ้ำโดยอัตโนมัติ
- ปรับเปลี่ยนได้: เซตสามารถเพิ่มหรือลบข้อมูลได้ภายหลัง
- ตรวจสอบสมาชิกได้ง่าย: มีฟังก์ชันสำหรับตรวจสอบว่าสมาชิกอยู่ในเซตหรือไม่
- การดำเนินการเชิงคณิตศาสตร์: รองรับบูนเดียน อินเทอร์เซกชัน และดิฟเฟอเรนซ์
- ไม่รองรับการเข้าถึงด้วยดัชนี: ไม่สามารถเข้าถึงข้อมูลด้วยดัชนีหรือตัดช่วงได้เหมือนลิสต์
- ขนาดเปลี่ยนแปลงได้: จำนวนสมาชิกในเซตสามารถเปลี่ยนแปลงได้
- สมาชิกต้องแข็งได้: สมาชิกของเซตต้องเป็นชนิดข้อมูลที่ไม่เปลี่ยนแปลง เช่น string, number, หรือ tuple
- การใช้งาน: ใช้เมื่อต้องการข้อมูลที่ไม่ซ้ำกัน การลบค่าซ้ำ และการดำเนินการทางเซต
- การสร้าง: สร้างได้ด้วย {} หรือ set() จากลิสต์หรือทูเพิล

CREATING SET

```
1 # Creating a set
2 fruits = {"apple", "banana", "cherry", "apple"}
3 print(fruits) # Output: {'apple', 'banana', 'cherry'} (
4   duplicates are removed)
5
5 # Creating a set using the set() function
6 numbers = set([1, 2, 3, 1, 2, 4])
7 print(numbers) # Output: {1, 2, 3, 4}
```

Listing 7.1: Creating and Accessing Sets

SET MEMBER MANAGEMENT

ฟังก์ชันสำหรับการจัดการสมาชิกในเซต

- `add()`: เพิ่มสมาชิกหนึ่งรายการ
- `update()`: เพิ่มหลายรายการพร้อมกัน
- `remove()`: ลบสมาชิก หากไม่พบจะเกิดข้อผิดพลาด
- `discard()`: ลบสมาชิก หากไม่พบจะไม่เกิดข้อผิดพลาด
- `pop()`: ลบและคืนค่ารายการหนึ่งแบบสุ่ม
- `clear()`: ลบสมาชิกทั้งหมดในเซต

SET MEMBER MANAGEMENT

- ฟังก์ชัน `len`: ใช้คืนค่าจำนวนสมาชิกในเซต
- เซต เป็นวัตถุที่สามารถเปลี่ยนแปลงได้ (**mutable**)
- เมธอด `add`: ใช้เพิ่มสมาชิกหนึ่งตัวลงในเซต
- เมธอด `update`: ใช้เพิ่มกลุ่มของสมาชิกหลายตัวลงในเซต
- อาร์กิวเม้นต์ต้องเป็นลำดับ (**sequence**) ที่มีสมาชิกแบบวนซ้ำได้ (**iterable**) และสมาชิกแต่ละตัวจะถูกเพิ่มลงในเซต

SET MEMBER MANAGEMENT

เมธอด **remove** และ **discard**: ใช้ลบสมาชิกที่ระบุออกจากเซต

สมาชิกที่ต้องการลบจะถูกส่งเป็นอาร์กิว เมนต์ให้กับทั้งสอง เมธอด

ทั้งสอง เมธอดมีพุติกรรมต่างกัน เมื่อไม่พบสมาชิกที่ระบุในเซต

- เมธอด **remove** จะทำให้เกิดข้อยกเว้นประเภท **KeyError**
- เมธอด **discard** จะไม่ทำให้เกิดข้อยกเว้น
- เมธอด **clear**: ใช้ลบสมาชิกทั้งหมดออกจากเซต

SET MEMBER MANAGEMENT

```
1 # Adding and removing items
2 fruits = {"apple", "banana", "cherry"}
3
4 fruits.add("orange")
5 print(fruits) # Output: {'apple', 'banana', 'cherry', 'orange'}
6
7 fruits.remove("banana")
8 print(fruits) # Output: {'apple', 'cherry', 'orange'}
9
10 fruits.discard("grape")
11 print(fruits) # Output: {'apple', 'cherry', 'orange'} (no error
12     raised)
13
14 removed_item = fruits.pop()
15 print(removed_item)
16 print(fruits) # Output: {'cherry', 'orange'} (an arbitrary item
17     removed)
18
19 fruits.clear()
20 print(fruits) # Output: set() (empty set)
```

Listing 7.2: Adding and Removing Items to Set

SET OPERATIONS

ฟังก์ชันและสัญลักษณ์ของการดำเนินการเซต

- `union() |`: รวมสมาชิกของทั้งสองเซต
- `intersection() &`: หาสมาชิกที่มีร่วมกัน
- `difference() -`: หาสมาชิกที่มีในเซตแรกแต่ไม่มีในเซตที่สอง
- `symmetric_difference() ^` : หาสมาชิกที่อยู่ในเซตใดเซตหนึ่งเท่านั้น

SET OPERATIONS

```
1 set1 = {1, 2, 3}
2 set2 = {3, 4, 5}
3
4 # Union
5 print(set1.union(set2)) # Output: {1, 2, 3, 4, 5}
6
7 # Intersection
8 print(set1.intersection(set2)) # Output: {3}
9
10 # Difference
11 print(set1.difference(set2)) # Output: {1, 2}
12
13 # Symmetric Difference
14 print(set1.symmetric_difference(set2)) # Output: {1, 2, 4, 5}
```

Listing 7.3: Set Operations

SET OPERATIONS

```
1 set1 = {1, 2, 3, 4}
2 set2 = {3, 4, 5, 6}
3
4 # Union
5 union_set = set1 | set2
6 print("Union:", union_set) # Output: {1, 2, 3, 4, 5, 6}
7
8 # Intersection
9 intersection_set = set1 & set2
10 print("Intersection:", intersection_set) # Output: {3, 4}
11
12 # Difference
13 difference_set = set1 - set2
14 print("Difference:", difference_set) # Output: {1, 2}
15
16 # Symmetric Difference
17 sym_diff_set = set1 ^ set2
18 print("Symmetric Difference:", sym_diff_set)
19 # Output: {1, 2, 5, 6}
```

Listing 7.4: Set Symbol Operations

SET OPERATION

A B A.union(B)	Returns a set which is the union of sets A and B .
A = B A.update(B)	Adds all elements of array B to the set A .
A & B A.intersection(B)	Returns a set which is the intersection of sets A and B .
A &= B A.intersection_update(B)	Leaves in the set A only items that belong to the set B .
A - B A.difference(B)	Returns the set difference of A and B (the elements included in A , but not included in B).
A -= B A.difference_update(B)	Removes all elements of B from the set A .
A ^ B A.symmetric_difference(B)	Returns the symmetric difference of sets A and B (the elements belonging to either A or B , but not to both sets simultaneously).
A ^= B A.symmetric_difference_update(B)	Writes in A the symmetric difference of sets A and B .
A <= B A.issubset(B)	Returns true if A is a subset of B .
A >= B A.issuperset(B)	Returns true if B is a subset of A .
A < B	Equivalent to A <= B and A != B
A > B	Equivalent to A >= B and A != B

SETINTRO.PY

```
1  setA = {1, 2, 3, 4}
2  setB = set([8, 9, 10])
3
4  setA.add(5)
5  setB.update([6, 7])
6  Uset = setA | setB
7  print(Uset)
8  print(len(Uset))
9
10 setB.update('ABCD')
11 setA.update([6, 7, 8])
12 print(setB)
13
14 print(setA.intersection(setB))
15 print(setA ^ setB)
16
17 setB.remove('B')
18 setB.discard(10)
19 print(setB)
20 print(setA.clear())
21 for val in Uset:
22     | | print(val)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
10
{'B', 6, 7, 8, 9, 10, 'C', 'D', 'A'}
{8, 6, 7}
{'B', 1, 2, 3, 4, 9, 10, 'C', 5, 'D', 'A'}
{6, 7, 8, 9, 'C', 'D', 'A'}
None
1
2
3
4
5
6
7
8
9
10
```

UPDATE SET OPERATION

ในภาษา Python เราสามารถอัปเดตเซตได้โดยตรงด้วยการใช้โอเปอเรเตอร์พิเศษ ซึ่งจะปรับปรุงค่าในเซตโดยอิงจากค่าของเซตอื่น โดยโอเปอเรเตอร์เหล่านี้ได้แก่ `&=`, `-=`, และ `^=` ซึ่งมีความหมายดังนี้:

- `&=` (Intersection Update): เก็บไว้เฉพาะสมาชิกที่มีอยู่ในทั้งสองเซต
- `-=` (Difference Update): ลบสมาชิกที่มีอยู่ในเซตอีกชุดออกจากเซตเดิม
- `^=` (Symmetric Difference Update): เก็บไว้เฉพาะสมาชิกที่อยู่ในเซตใดเซตหนึ่งเท่านั้น แต่ไม่ในทั้งสองไม่ได้

UPDATE SET OPERATION

```
1 # Initial sets
2 set1 = {1, 2, 3, 4, 5}
3 set2 = {4, 5, 6, 7}
4
5 # &= Intersection Update: set1 will be updated only to include elements
   present in both sets
6 set1 &= set2
7 print("After &= operation:", set1) # Output: {4, 5}
8
9 # Resetting set1 to its original value
10 set1 = {1, 2, 3, 4, 5}
11
12 # -= Difference Update: set1 will be updated to remove elements also
   present in set2
13 set1 -= set2
14 print("After -= operation:", set1) # Output: {1, 2, 3}
15
16 # Resetting set1 to its original value
17 set1 = {1, 2, 3, 4, 5}
18
19 # ^= Symmetric Difference Update: set1 will be updated to keep elements in
   either set but not in both
20 set1 ^= set2
21 print("After ^= operation:", set1) # Output: {1, 2, 3, 6, 7}
```

Listing 7.5: Demonstration of `&=` `-=` `^=` operations in Python sets

SUBSET AND SUPERSET

โอเปอเรเตอร์เหล่านี้ใช้สำหรับเปรียบเทียบความสัมพันธ์ระหว่างเซตสองชุดว่ามีลักษณะเป็นซับเซต ชูเปอร์เซต หรือเท่ากัน ดังนี้:

- \geq (ชูเปอร์เซต): ตรวจสอบว่าเซตหนึ่งเป็นชูเปอร์เซตของอีกเซตหรือไม่ (มีสมาชิกทั้งหมดของอีกเซต)
- \leq (ซับเซต): ตรวจสอบว่าเซตหนึ่งเป็นซับเซตของอีกเซตหรือไม่ (สมาชิกทั้งหมดอยู่ในอีกเซต)
- $>$ (ชูเปอร์เซตแท้): เป็นชูเปอร์เซตและมีสมาชิกมากกว่าอีกเซต
- $<$ (ซับเซตแท้): เป็นซับเซตและมีสมาชิกน้อยกว่าอีกเซต
- $=$ (เท่ากัน): ตรวจสอบว่าเซตสองชุดมีสมาชิกเท่ากันทุกตัว

SUBSET AND SUPERSET

```
1 # Define two sets
2 set_a = {1, 2, 3, 4}
3 set_b = {2, 3}
4 set_c = {1, 2, 3, 4}
5 set_d = {1, 2, 3, 4, 5}
6
7 # Superset and Subset
8 print("Is set_a a superset of set_b?:", set_a >= set_b) # Output: True
9 print("Is set_b a subset of set_a?:", set_b <= set_a) # Output: True
10
11 # Proper Superset and Proper Subset
12 print("Is set_a a proper superset of set_b?:", set_a > set_b) # Output:
13 # True
13 print("Is set_b a proper subset of set_a?:", set_b < set_a) # Output:
14 # True
14
15 # Equal Sets
16 print("Are set_a and set_c equal?:", set_a == set_c) # Output: True
17
18 # Related but not equal (one is a subset but not equal)
19 print("Is set_b a subset of set_d and not equal?:", set_b <= set_d and
set_b != set_d) # Output: True
```

Listing 7.6: Demonstration of Set Operations in Python

THE POWER OF USING SETS WITH LISTS

```
1 def remove_duplicates(lst):  
2     return list(set(lst))  
3  
4 numbers = [1, 2, 3, 1, 2, 4, 5, 6, 5, 4, 3]  
5 print(remove_duplicates(numbers))  
6 # Output: [1, 2, 3, 4, 5, 6]
```

THE POWER OF USING SETS WITH LISTS

```
1 # Attendance records for a week (each list represents a day's attendance)
2 attendance_week = [
3     ["Alice", "Bob", "Charlie", "David"],    # Day 1
4     ["Alice", "Charlie", "David"],           # Day 2
5     ["Alice", "Bob", "David"],               # Day 3
6     ["Alice", "David", "Eve"],              # Day 4
7     ["Bob", "Charlie", "David"]            # Day 5
8 ]
9
10 # 1. Find the set of students who were present every day.
11 # 2. Determine the set of students who were absent at least one day.
12 # 3. Create a list of students who were present on the first day but absent on the last day.
13 # 4. Calculate the total number of unique students who attended at least one day.
14
```

THE POWER OF USING SETS WITH LISTS

```
14  
15 # Convert each day's attendance list into a set  
16 attendance_sets = [set(day) for day in attendance_week]  
17 print(attendance_sets)  
18
```

```
attendance_sets = [  
    {"Alice", "Bob", "Charlie"},  
    {"Alice", "David", "Charlie"},  
    {"Bob", "David", "Eve"}  
]
```

THE POWER OF USING SETS WITH LISTS

```
18
19 # 1. Find the set of students who were present every day.
20 present_every_day = set.intersection(*attendance_sets)
21 print("Present every day:", present_every_day)
22 # Output: {'David'}
23
24 # 2. Determine the set of students who were absent at least one day.
25 all_students = set.union(*attendance_sets)
26 absent_at_least_one_day = all_students - present_every_day
27 print("Absent at least one day:", absent_at_least_one_day)
28 # Output: {'Alice', 'Charlie', 'Bob', 'Eve'}
29
```

THE POWER OF USING SETS WITH LISTS

```
29
30 # 3. Create a list of students who were present on the first day but absent on the last day.
31 first_day_present = attendance_sets[0]
32 last_day_present = attendance_sets[-1]
33 first_day_but_not_last = list(first_day_present - last_day_present)
34 print("Present on first day but absent on last day:", first_day_but_not_last)
35 # Output: ['Alice']
36
37 # 4. Calculate the total number of unique students who attended at least one day.
38 unique_students_count = len(all_students)
39 print("Total unique students:", unique_students_count)
40 # Output: 5
```

EXERCISE-I

```
1 # Survey results (each list represents a participant's choices)
2 survey_results = [
3     ["Python", "JavaScript", "C++"],           # Participant 1
4     ["Python", "JavaScript", "C#"],            # Participant 2
5     ["Python", "Java"],                      # Participant 3
6     ["Python", "C++", "JavaScript"],          # Participant 4
7     ["Python", "JavaScript", "C++", "Java"],   # Participant 5
8 ]
9 # 1. Identify the languages that were chosen by all participants.
10 # 2. Find the languages that were only chosen by a single participant.
11 # 3. Determine the number of unique languages mentioned in the survey.
12 # 4. List the languages that were chosen by exactly two participants.
13 # 5. Find participants who have the exact same set of favorite languages.
14
15
```

EXERCISE-I

1. Languages chosen by all participants: {'Python'}
2. Languages only chosen by one participant: {'C#'}
3. Number of unique languages: 5
4. Languages chosen by exactly two participants: {'Java'}
5. Participants with the same set of languages: [[1, 4]]

EXERCISE-I

```
# 1. Languages chosen by all participants
choices_sets = [set(p) for p in survey_results]
common_languages = set.intersection(*choices_sets)
print("1. Languages chosen by all participants:", common_languages)
```

DICTIONARIES



Introduction

DICTIONARIES

- **ดิกชันนารี (Dictionary):** วัตถุที่ใช้เก็บกลุ่มของข้อมูล
แต่ละสมาชิกประกอบด้วยคีย์ (key) และค่า (value)
มักเรียกว่าเป็นการจับคู่ระหว่างคีย์กับค่า (mapping of key to value)
คีย์ต้องเป็นวัตถุที่ไม่สามารถเปลี่ยนแปลงได้ (immutable)
การดึงค่าที่ต้องการใช้คีย์ที่เชื่อมโยงกับค่านั้น
- **รูปแบบการสร้างดิกชันนารี:**
`dictionary = {key1:val1, key2:val2}`

ADD AND DELETE DICTIONARY

- ดิกชันนารีเป็นวัตถุที่สามารถเปลี่ยนแปลงได้ (mutable)

การเพิ่มคู่คีย์-ค่าใหม่: `dictionary[key] = value`

ถ้าคีย์มีอยู่แล้ว ในดิกชันนารี ค่าที่เชื่อมโยงกับคีย์นั้นจะถูกเปลี่ยน

- การลบคู่คีย์-ค่า: `del dictionary[key]`

ถ้าคีย์ไม่อยู่ในดิกชันนารี จะเกิดข้อข้อความประภาก `KeyError`

DICTIONARIES

แนวคิดหลักของ Dictionary:

- ไม่เรียงลำดับ (Unordered Collection): ไม่มีลำดับที่แน่นอนของสมาชิก
- คู่คีย์-ค่า (Key-Value Pairs): ข้อมูลถูกจัดเก็บเป็นคู่คีย์และค่าที่สัมพันธ์กัน
- คีย์ต้องไม่ซ้ำกัน (Unique Keys): คีย์แต่ละตัวต้องไม่ซ้ำ
- เปลี่ยนแปลงได้ (Mutable): สามารถแก้ไขหรือปรับเปลี่ยนได้หลังจากสร้างแล้ว
- ขนาดเปลี่ยนแปลงได้ (Dynamic Size): เพิ่มหรือลบคู่คีย์-ค่าได้ตามต้องการ
- เข้าถึงได้รวดเร็ว (Efficient Lookup): ค้นหาค่าได้อย่างรวดเร็วโดยใช้คีย์
- คีย์ต้อง hash ได้ (Hashable Keys): คีย์ต้องเป็นชนิดข้อมูลที่ไม่เปลี่ยนแปลง
- ค่ามีความยืดหยุ่น (Flexible Values): ค่าสามารถเป็นข้อมูลชนิดใดก็ได้
- มีเมธอดหลากหลาย (Comprehensive Methods): มีฟังก์ชันสำหรับเข้าถึง ตรวจสอบ และลบ ข้อมูล
- การใช้งาน (Use Cases): เหมาะสำหรับจัดเก็บข้อมูลที่มีป้ายกำกับหรือต้องการเชื่อมโยงเป็นคู่

CREATING AND ACCESSING DICTIONARIES

```
1 # Creating a dictionary using {}
2 student = {"name": "Alice", "age": 25, "grade": "A"}
3 print(student) # Output: {'name': 'Alice', 'age': 25, 'grade': 'A'}
```

Listing 7.7: Creating a dictionary using {}

```
1 # Creating a dictionary using dict() with keyword arguments
2 student = dict(name="Alice", age=25, grade="A")
3 print(student) # Output: {'name': 'Alice', 'age': 25, 'grade': 'A'}
4
5 # Creating a dictionary using dict() with a list of tuples
6 student = dict([("name", "Alice"), ("age", 25), ("grade", "A")])
7 print(student) # Output: {'name': 'Alice', 'age': 25, 'grade': 'A'}
```

Listing 7.8: Creating a dictionary using `dict()` with keyword arguments

CREATING AND ACCESSING DICTIONARIES

```
1 # Creating an empty dictionary
2 student = {}
3
4 # Adding key-value pairs using assignment
5 student["name"] = "Alice"
6 student["age"] = 25
7 student["grade"] = "A"
8 print(student) # Output: {'name': 'Alice', 'age': 25, 'grade': 'A'}
```

Listing 7.9: Creating an empty dictionary and adding key-value pairs using `=`

CREATING AND ACCESSING DICTIONARIES

```
1 # Creating a dictionary
2 student = {"name": "Alice", "age": 25, "grade": "A"}
3
4 # Accessing values by key
5 print(student["name"]) # Output: Alice
6 print(student["age"]) # Output: 25
7 print(student["grade"]) # Output: A
```

Listing 7.10: Creating and Accessing Dictionaries

INTRODICT.PY

```
1 phonebook = {'Anirach': '777-1111', 'Mickey': '777-2222', 'Donald': '777-3333'}
2
3 # display dictionary contents
4 print(phonebook)
5
6 # retive specic key value
7 print(phonebook['Mickey'])
8 print(phonebook.get('Donald'))
9
10 key = 'Pluto'
11 if key in phonebook:
12     print(phonebook['Pluto'])
13 else:
14     print(key + ' not in phonebook')
15
16 phonebook['Simpson'] = '777-4567'
17 phonebook['Pluto'] = '777-4444'
18 phonebook['Mickey'] = '777-2122'
19 print(phonebook)          {'Anirach': '777-1111', 'Mickey': '777-2222', 'Donald': '777-3333'}
20                                         777-2222
21                                         777-3333
22 del phonebook['Simpson']      Pluto not in phonebook
23                                         {'Anirach': '777-1111', 'Mickey': '777-2122', 'Donald': '777-3333', 'Simpson': '777-4567', 'Pluto': '777-4444'}
24 print(phonebook)              {'Anirach': '777-1111', 'Mickey': '777-2122', 'Donald': '777-3333', 'Pluto': '777-4444'}
```

ACCESSING DICTIONARIES WITH LOOP

Example 1: Iterating Over Keys and Accessing Values

This example demonstrates looping through the keys and accessing the corresponding values.

```
1 student = {"name": "Alice", "age": 25, "grade": "A", "major": "Computer Science"}  
2  
3 for key in student:  
4     print(f"{key}: {student[key]}")  
5 # Output:  
6 # name: Alice  
7 # age: 25  
8 # grade: A  
9 # major: Computer Science
```

Listing 7.11: Iterating Over Keys and Accessing Values

ACCESSING DICTIONARIES WITH LOOP

Example 2: Iterating Over Values

This example shows how to loop through just the values of the dictionary.

```
1 student = {"name": "Alice", "age": 25, "grade": "A", "major": "Computer Science"}  
2  
3 for value in student.values():  
4     print(value)  
5 # Output:  
6 # Alice  
7 # 25  
8 # A  
9 # Computer Science
```

Listing 7.12: Iterating Over Values

ACCESSING DICTIONARIES WITH LOOP

Example 3: Iterating Over Key-Value Pairs

This example illustrates looping through keys and values simultaneously using the `.items()` method.

```
1 student = {"name": "Alice", "age": 25, "grade": "A", "major": "Computer Science"}  
2  
3 for key, value in student.items():  
4     print(f"{key}: {value}")  
5 # Output:  
6 # name: Alice  
7 # age: 25  
8 # grade: A  
9 # major: Computer Science
```

Listing 7.13: Iterating Over Key-Value Pairs

MODIFYING DICTIONARIES

```
1 # Creating a dictionary
2 student = {"name": "Alice", "age": 25, "grade": "A"}
3
4 # Adding and updating values
5 student["age"] = 26
6 student["major"] = "Computer Science"
7 print(student) # Output: {'name': 'Alice', 'age': 26, 'grade': 'A',
8 #                         'major': 'Computer Science'}
9
10 # Removing a key-value pair using del
11 del student["grade"]
12 print(student) # Output: {'name': 'Alice', 'age': 26, 'major': 'Computer Science'}
13
14 # Removing a key-value pair using pop and getting the removed
15 # value
16 removed_major = student.pop("major")
17 print(removed_major) # Output: 'Computer Science'
18 print(student) # Output: {'name': 'Alice', 'age': 26}
```

Listing 7.8: Example: Using `del` and `pop()` in a Dictionary

FORDICT.PY

```
1 phonebook = {'Anirach': '777-1111', 'Mickey': '777-2222', 'Donald': '777-3333'}
2
3 phonebook['Bart'] = [1, 3, 5]
4
5 elements = len(phonebook)
6 print('There are ', elements, ' names in phonebook')
7
8 for key in phonebook:
9     print(key, 'phone number is: ', phonebook[key])
10
11 phonebook['Bart'][1] = 9
12 print(phonebook)
```

```
There are 4 names in phonebook
Anirach phone number is: 777-1111
Mickey phone number is: 777-2222
Donald phone number is: 777-3333
Bart phone number is: [1, 3, 5]
{'Anirach': '777-1111', 'Mickey': '777-2222', 'Donald': '777-3333', 'Bart': [1, 9, 5]}
```

DICTIONARY METHODS

เมธอดของ Dictionary

- `keys()`: คืนค่าออบเจกต์ที่เป็นมุ่งมองของคีย์ทั้งหมดใน Dictionary
- `values()`: คืนค่าออบเจกต์ที่เป็นมุ่งมองของค่าทั้งหมดใน Dictionary
- `items()`: คืนค่าออบเจกต์ที่เป็นมุ่งมองของคู่คีย์-ค่าทั้งหมดใน Dictionary
- `get()`: คืนค่าที่ตรงกับคีย์ที่ระบุ ถ้าคีย์นั้นมีอยู่ใน Dictionary
- `pop()`: ลบและคืนค่าที่ตรงกับคีย์ที่ระบุ
- `popitem()`: ลบและคืนคู่คีย์-ค่าล่าสุดจาก Dictionary
- `clear()`: ลบคู่คีย์-ค่าทั้งหมดใน Dictionary

DICTIONARY METHODS

```
1 # Creating a dictionary
2 student = {'name': 'Alice', 'age': 26, 'major': 'Computer Science'
3
4 # Dictionary methods
5 print(student.keys())    # Output: dict_keys(['name', 'age', 'major'])
6 print(student.values())  # Output: dict_values(['Alice', 26, 'Computer Science'])
7 print(student.items())   # Output: dict_items([('name', 'Alice'), ('age', 26), ('major', 'Computer Science')])
8
9 # Using get() method
10 print(student.get("name")) # Output: Alice
11 print(student.get("grade", "Not Found")) # Output: Not Found
12
13 # Using pop() method
14 major = student.pop("major")
15 print(major) # Output: Computer Science
16 print(student) # Output: {'name': 'Alice', 'age': 26}
17
18 # Using popitem() method
19 last_item = student.popitem()
20 print(last_item) # Output: ('age', 26)
21 print(student) # Output: {'name': 'Alice'}
22
23 # Using clear() method
24 student.clear()
25 print(student) # Output: {}
```

Listing 7.15: Dictionary Methods

DICTMETHODS.PY

EXERCISE-II

```
1 # Sample data structure for employee performance
2 performance_data = {
3     "Sales": {
4         "Alice": [80, 85, 88, 90],
5         "Bob": [70, 75, 78, 80],
6         "Charlie": [60, 65, 70, 72]
7     },
8     "Engineering": {
9         "David": [90, 92, 94, 95],
10        "Eve": [85, 88, 87, 90],
11        "Frank": [88, 87, 86, 85]
12    },
13    "HR": {
14        "Grace": [70, 72, 74, 76],
15        "Heidi": [65, 68, 70, 73],
16        "Ivan": [60, 62, 64, 66]
17    }
18 }
19
20 # 1. Calculate the average performance score for each employee.
21 # 2. Identify the top performer in each department based on their average score.
22 # 3. Determine the department with the highest average performance score.
23 # 4. Find employees who have shown continuous improvement.
24 # 5. Generate a summary report.
```

EXERCISE-II

```
Average Performance Scores: {'Sales': {'Alice': 85.75, 'Bob': 75.75, 'Charlie': 66.75}, 'Engineering': {'David': 92.75, 'Eve': 87.5, 'Frank': 86.5}, 'HR': {'Grace': 73.0, 'Heidi': 69.0, 'Ivan': 63.0}}
```

```
Top Performers: {'Sales': ('Alice', 85.75), 'Engineering': ('David', 92.75), 'HR': ('Grace', 73.0)}
```

```
Best Department: Engineering 88.91666666666667
```

```
Continuous Improvers: {'Sales': ['Alice', 'Bob', 'Charlie'], 'Engineering': ['David'], 'HR': ['Grace', 'Heidi', 'Ivan']}
```

EXERCISE-II

Summary Report:

Department: Sales

Alice: Average Score = 85.75

Bob: Average Score = 75.75

Charlie: Average Score = 66.75

Top Performer: Alice with Average Score = 85.75

Department: Engineering

David: Average Score = 92.75

Eve: Average Score = 87.50

Frank: Average Score = 86.50

Top Performer: David with Average Score = 92.75

Department: HR

Grace: Average Score = 73.00

Heidi: Average Score = 69.00

Ivan: Average Score = 63.00

Top Performer: Grace with Average Score = 73.00

Best Department: Engineering with Average Score = 88.92

EXERCISE-II

```
# 1. Calculate the average performance score for each employee
average_scores = {}
for department, employees in performance_data.items():
    average_scores[department] = {}
    for employee, scores in employees.items():
        average = sum(scores) / len(scores)
        average_scores[department][employee] = average
```

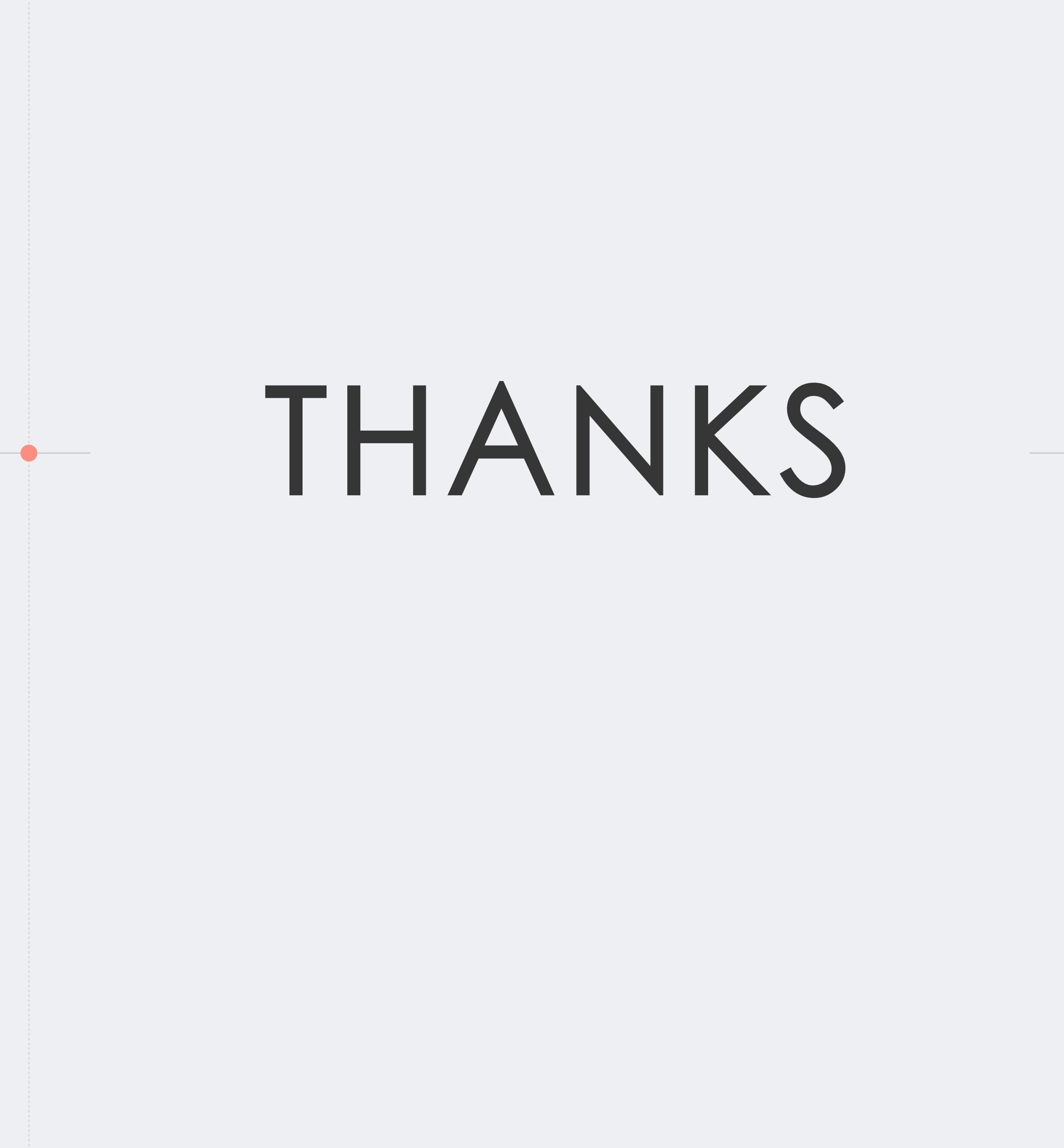
SUMMARY

Dictionaries, including:

- Creating dictionaries
- Inserting, retrieving, adding, and deleting key-value pairs
- for loops and in and not in operators
- Dictionary methods

Sets:

- Creating sets
- Adding elements to and removing elements from sets
- Finding set union, intersection, difference, and symmetric difference
- Finding subsets and supersets



THANKS

WORD OF THE WISE



Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program.

— *Linus Torvalds* —

AZ QUOTES