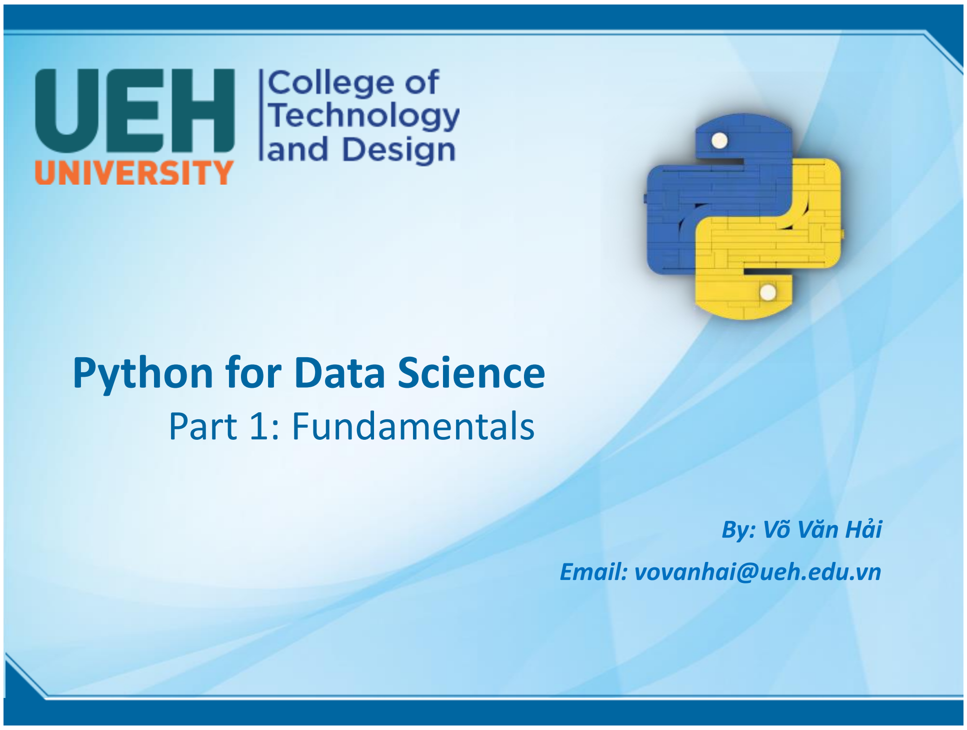


UEH UNIVERSITY | College of Technology and Design

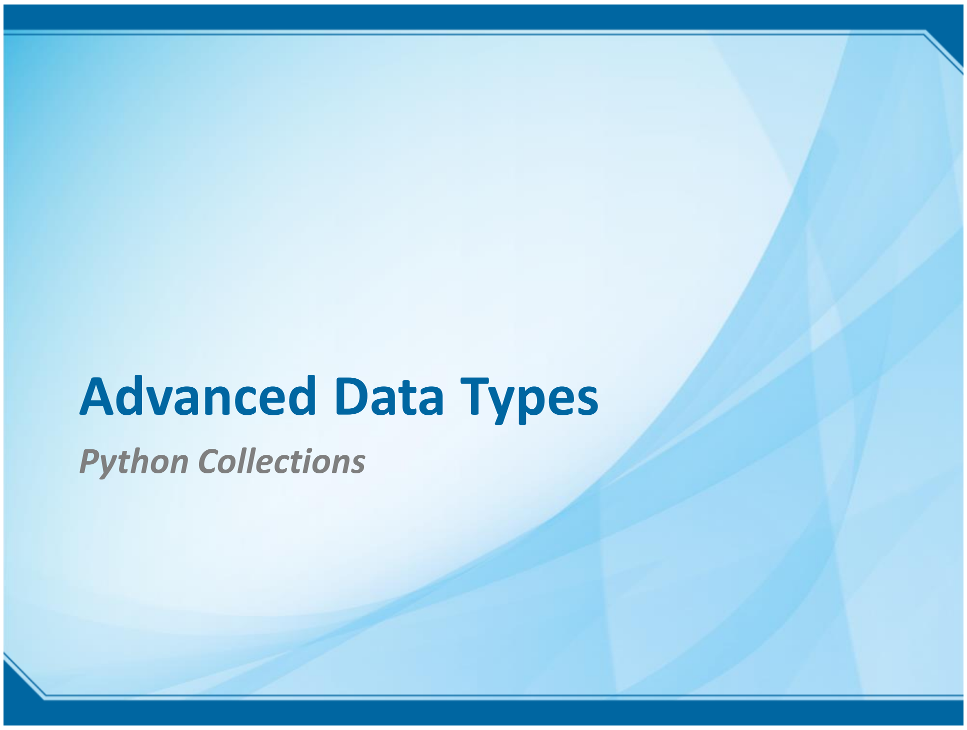
Python for Data Science

Part 1: Fundamentals

By: Võ Văn Hải

Email: vovanhai@ueh.edu.vn

1



Advanced Data Types

Python Collections

2

Python Collections

Introduction

- There are four collection data types in the Python programming language:
  - **List** is a collection which is ordered and changeable. Allows duplicate members.
  - **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
  - **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
  - **Dictionary** is a collection which is ordered** and changeable. No duplicate members.
    - **Set items are unchangeable, but you can remove and/or add items whenever you like.*
    - ***As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.*

Python Collections

Python List

List

Introduction

- Lists are used to store multiple items in a single variable.
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
- Lists are created using square brackets:

```
fruit_list = ["apple", "banana", "cherry"]
```

- Characteristics
  - **Ordered** - They maintain the order of elements.
  - **Mutable** - Items can be changed after creation.
  - **Allow duplicates** - They can contain duplicate values.

```
fruit_list = ["apple", "banana", "cherry", "apple", "cherry"]
fruit_list.append("banana")
fruit_list.remove("cherry")

print(fruit_list)  # ->['apple', 'banana', 'apple', 'cherry', 'banana']
```

Use the len() function to determine the list's number of items.

```
print(len(fruit_list))
```

List

Create list

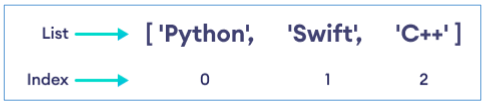
- We can use the built-in list() function to convert other iterable (strings, dictionaries, tuples, etc.) to a list.

```
x = "axz"
# convert to list
result = list(x)
print(result)  # ['a', 'x', 'z']
```

```
# note the double round-brackets
fruit_list = list(("apple", "banana", "cherry"))
print(fruit_list)  # ['apple', 'banana', 'cherry']
```

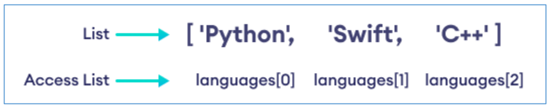
List

Access List Elements



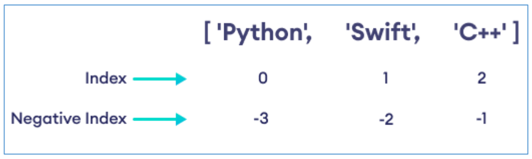
Index of List Elements

- We use these index numbers to access list items.



```
languages = ['Python', 'Swift', 'C++']
# Access the first element
print(languages[0])  # Python
# Access the third element
print(languages[2])  # C++
```

- Negative Indexing in Python
  - Python also supports negative indexing. The index of the last element is -1, the second-last element is -2, and so on.



```
languages = ['Python', 'Swift', 'C++']
# Access item at index 0
print(languages[-1])  # C++
# Access item at index 2
print(languages[-3])  # Python
```

List

Slicing of a list

- It is possible to access a section of items from the list using the slicing operator :

```
my_list = ['p', 'r', 'o', 'g', 'r', 'a', 'm']

# items from index 2 to index 4
print(my_list[2:5])  # ->['o', 'g', 'r']

# items from index 5 to end
print(my_list[5:])  # ->['a', 'm']

# items beginning to end
print(my_list[:])  # ->['p', 'r', 'o', 'g', 'r', 'a', 'm']
```

***Note**: If the specified index does not exist in a list, Python throws the IndexError exception.*

List

Modify a list

```
fruits = ['apple', 'banana', 'orange']
# using append method to add item add the end of list
fruits.append('cherry')  # ['apple', 'banana', 'orange', 'cherry']

# insert 'cherry' at index 2
fruits.insert(2, 'grape')  # ['apple', 'banana', 'grape', 'orange', 'cherry']

# changing the third item to 'mango'
fruits[2] = 'mongo'  # ->['apple', 'banana', 'mango', 'orange', 'cherry']

# remove 'banana' from the list (first occurrence)
fruits.remove("banana")  # ->['apple', 'mango', 'orange', 'cherry']

# deleting the second item
del fruits[1]  # ->['apple', 'orange', 'cherry']

# deleting items from index 0 to index 2
del fruits[0:2]  # ->['cherry']
```

The list slice technique can be used to modify list items.

Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon":

```
fruits = ["apple", "banana", "cherry", "orange",
"kiwi", "mango"]
fruits[1:3] = ["blackcurrant", "watermelon"]
```

```
# the extend() method can be used to add elements to a list from other
iterables.
odd_numbers, even_numbers = [1, 3, 5], [2, 4, 6]
# adding elements of one list to another
odd_numbers.extend(even_numbers)  # ->[1, 3, 5, 2, 4, 6]
print('Updated Numbers:', odd_numbers)
```

List

Sort lists

- Sort the list alphanumerically

```
fruits = ["orange", "mango", "kiwi", "pineapple", "banana"]
fruits.sort()  # ->['banana', 'kiwi', 'mango', 'orange', 'pineapple']
```

- Sort descending (use the keyword argument reverse = True)

```
fruits = ["orange", "mango", "kiwi", "pineapple", "banana"]
fruits.sort(reverse=True)  # ->['pineapple', 'orange', 'mango', 'kiwi', 'banana']
```

- Case Insensitive Sort

```
fruits = ["orange", "Mango", "kiwi", "Pineapple", "banana"]
fruits.sort()  # ->['Mango', 'Pineapple', 'banana', 'kiwi', 'orange']
fruits.sort(key=str.lower)  # ->['banana', 'kiwi', 'Mango', 'orange', 'Pineapple']
fruits.sort(key=len)  # ->['kiwi', 'Mango', 'orange', 'banana', 'Pineapple']
```

- The **sort()** *vs.* **sorted()** methods:
  - The sort() method doesn't return anything, it modifies the original list (i.e. sorts in-place). If you don't want to modify the original list, use sorted() function. It returns a sorted copy of the list.

```
new_list = sorted(fruits)
print(new_list)
```

List

Copy Lists

- DO NOT copy a list by using the assign operator

  list2 = list1,

- Cause: *list2* will only be a *reference* to *list1*, and changes made in *list1* will automatically also be made in *list2*.

- Make a copy of a list
  - With the *copy()* method
  - By using the built-in method *list()*
  - By using the **:** (slice) operator.

```
fruits = ["apple", "banana", "cherry"]
new_list_1 = fruits.copy()
new_list_2 = list(fruits)
new_list_3 = fruits[:]
```

List

List methods

- Iterating tthrough a List

```
fruits = ['apple', 'banana', 'orange']
# iterate through the list
for fruit in fruits:
    print(fruit)
```

```
fruits = ["apple", "banana", "cherry"]
# looping using list comprehension
[print(x) for x in fruits]
```

- List methods

Method	Description
append()	Adds an item to the end of the list
extend()	Adds items of lists and other iterables to the end of the list
insert()	Inserts an item at the specified index
remove()	Removes item present at the given index
pop()	Returns and removes item present at the given index
clear()	Removes all items from the list
index()	Returns the index of the first matched item
count()	Returns the count of the specified item in the list
sort()	Sorts the list in ascending/descending order
reverse()	Reverses the item of the list
copy()	Returns the shallow copy of the list

List

Exercises

13

Python Collections

Python Tuple

14

Tuple

Introduction

- Tuples are used to store multiple items in a single variable.
- A tuple is a collection which is ordered and unchangeable.
- A tuple can be created by
  - placing items inside parentheses ().

```
fruits = ("orange", "Mango", "kiwi", "Pineapple", "banana")
```

  - using a tuple() constructor.

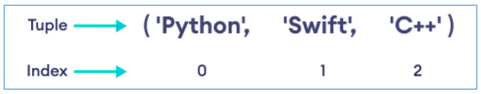
```
tuple_constructor = tuple(('Jack', 'Maria', 'David'))
```

- Tuple Characteristics
  - **Ordered** - They maintain the order of elements.
  - **Immutable** - They cannot be changed after creation (ie. Unchangeable).
  - **Allow duplicates** - They can contain duplicate values.

Tuple

Access Tuple Items

- Each item in a tuple is associated with a number, known as a index.
  - The index always starts from 0, meaning the first item of a tuple is at index 0, the second item is at index 1, and so on.



```
languages = ('Python', 'Swift', 'C++')
print(languages[1])
```

- Specify negative indexes if you want to start the search from the end of the tuple:

```
fruits = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
# returns the items from index -4 (included) to index -1 (excluded)
print(fruits[-4:-1])  # ->('orange', 'kiwi', 'melon')
```

- Check if an Item Exists in the Tuple

```
print('grape' in fruits)  # False
print('cherry' in fruits)  # True
```

Tuple

Update Tuple

- Tuples are immutable (unchangeable)→ cannot change the values
- We can convert the tuple into a list, change the list, and convert the list back into a tuple. ☺

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)  # -> ('apple', 'kiwi', 'cherry')
```

```
x = ("apple", "banana", "cherry")
y = list(x)
y.remove('apple')
x = tuple(y)
print(x)  # ->('banana', 'cherry')
```

- It is allowed to add tuples to tuples

```
fruits = ("apple", "banana", "cherry")
new_fruits = ("orange", "kiwi", "melon")
fruits += new_fruits  # ->('apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon')
```

- Iterate through the tuple

```
fruits = ('apple','banana','orange')
for fruit in fruits:
   print(fruit)
```

Tuple

Unpacking a Tuple

- In Python, there is a very powerful tuple assignment feature that assigns the right-hand side of values into the left-hand side.
- In another way, it is called unpacking of a tuple of values into a variable.
- In packing, we put values into a new tuple while in unpacking we extract those values into a single variable.

```
characters = ("Aragorn", "Gimli", "Legolas")   # python tuple packing line
(human, dwarf, elf) = characters   # python tuple unpacking line
print("Human :", human)
print("Dwarf :", dwarf)
print("Elf :", elf)
```

```
characters = ("Aragorn", "Gimli", "Legolas", "Elrond", "Galadriel")
(human, dwarf, *elf) = characters
print("Human :", human)
print("Dwarf :", dwarf)
print("Elf :", elf)
```

Tuple

Tuple methods

- Check if an Item Exists in the Tuple

```
colors = ('red', 'orange', 'blue')
print('yellow' in colors)    # False
print('red' in colors)       # True
```

- Deleting the tuple

```
animals = ('dog', 'cat', 'rat')
del animals
```

- Get the index of an item on Tuple

```
vowels = ('a', 'e', 'i', 'o', 'u')
index = vowels.index('e')
```

- Count the number of times the specified element appears in the tuple.

```
vowels = ('a', 'e', 'i', 'o', 'i', 'u')
# counts the number of i's in the tuple
count = vowels.count('i') # -->2
```

Tuple

Exercises

- Write a Python program
  - to create a tuple of numbers and print one item.
  - to unpack a tuple into several variables.
  - to add an item to a tuple.
  - to find the index of an item in a tuple.
  - to find the repeated items of a tuple

Python Collections

Python Set

Python Set

Introduction

- A Set in Python programming is an unordered collection data type that is iterable and has no duplicate elements.
- sets are created by placing all the elements inside curly braces {}, separated by commas.

```
# create a set of integer type
student_id = {112, 114, 116, 118, 115}
# create a set of string type
vowel_letters = {'a', 'e', 'i', 'o', 'u'}
# create a set of mixed data types
mixed_set = {'Hello', 101, -2, 'Bye'}
```

Create an Empty Set

```
# create an empty set
empty_set = set()  # type(empty_set)==> <class 'set'>

# create an empty dictionary
empty_dictionary = {} # type(empty_dictionary)==><class 'dict'>
```

- Typecasting list to set

```
# typecasting list to set
my_set = set(["a", "b", "c"])
```

Duplicate items in Set

```
numbers = {2, 4, 6, 6, 2, 8}
print(numbers)  # ==> {8, 2, 4, 6}
```

- Immutable set:

```
my_set = {"university", "economic", "HCMC"}
# Adding element to the set
my_set.add("CTD")
# values of a set cannot be changed
my_set[1] = "Academy" # -->TypeError: 'set' object does not support item assignment
```

Python Set

Update Python Set

- Update Python Set
  - The update() method is used to update the set with items other collection types (lists, tuples, sets, etc).

```
companies = {'Lacoste', 'Ralph Lauren'}
tech_companies = ['apple', 'google', 'apple']
companies.update(tech_companies)
```

- Remove an Element from a Set
  - The discard() method is used to remove the specified element from a set.

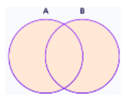
```
languages = {'Swift', 'Java', 'Python'}
print('Initial Set:', languages)
# remove 'Java' from a set
removedValue = languages.discard('Java')
print('Set after remove():', languages)  # -->{'Python', 'Swift'}
```

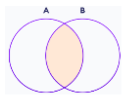
- Frozen sets in Python are immutable objects that only support methods and operators that produce a result without affecting the frozen set or sets to which they are applied.

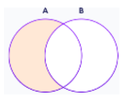
```
fruits = frozenset(["apple", "banana", "orange"])
fruits.append("pink")  # -->AttributeError: 'frozenset' object has no attribute 'append'
```

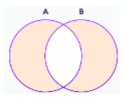
Python Set

Python Set Operations









```
A, B = {1, 3, 5}, {1, 2, 3}

print('Union using |:', A | B)
print('Union using union():', A.union(B))
```

The union of two sets A and B includes all the elements of sets A and B.

The intersection of two sets A and B include the common elements between set A and B.

```
print('Intersection using &:', A & B)
print('Intersection using intersection():', A.intersection(B))
```

The difference between two sets A and B include elements of set A that are not present on set B.

```
print('Difference using &:', A - B)
print('Difference using difference():', A.difference(B))
```

The symmetric difference between two sets A and B includes all elements of A and B without the common elements.

```
print('Symmetric Difference using ^:', A ^ B)
print('Symmetric Difference using symmetric_difference():', A.symmetric_difference(B))
```

Check if two sets are equal

```
A, B = {1, 3, 5}, {5, 3, 1}
if A == B:
    print('Set A and Set B are equal')
else:
    print('Set A and Set B are not equal')
```

Python Set

Python Set methods

Method	Description
add()	Adds an element to the set
clear()	Removes all elements from the set
copy()	Returns a copy of the set
difference()	Returns the difference of two or more sets as a new set
difference_update()	Removes all elements of another set from this set
discard()	Removes an element from the set if it is a member. (Do nothing if the element is not in set)
intersection()	Returns the intersection of two sets as a new set
intersection_update()	Updates the set with the intersection of itself and another
Isdisjoint()	Returns True if two sets have a null intersection

Method	Description
issubset()	Returns True if another set contains this set
issuperset()	Returns True if this set contains another set
pop()	Removes and returns an arbitrary set element. Raises KeyError if the set is empty
remove()	Removes an element from the set. If the element is not a member, raises a KeyError
symmetric_difference()	Returns the symmetric difference of two sets as a new set
symmetric_difference_update()	Updates a set with the symmetric difference of itself and another
union()	Returns the union of sets in a new set
update()	Updates the set with the union of itself and others

Python Set

Exercises

1. Write a Python program to find the maximum and minimum values in a set.
2. Write a Python program to check if a given value is present in a set or not.
3. Write a Python program to check if two given sets have no elements in common.
4. Write a Python program to find all the unique words and count the frequency of occurrence from a given list of strings. Use Python set data type.
5. Given two sets of numbers, write a Python program to find the missing numbers in the second set as compared to the first and vice versa. Use the Python set.

Python Collections

Python Dictionary

Python Dictionary

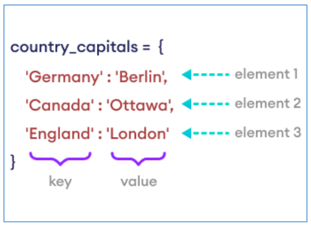
Introduction

- A Python dictionary is a data structure that stores the value in key:value pairs.

```
dict_var = { key1: value1’, key2: ‘value2’, ...}
```

- Python dictionaries are ordered and can not contain duplicate keys.

```
# creating a dictionary
country_capitals = {
   "Germany": "Berlin",
   "Canada": "Ottawa",
   "England": "London"
}
# printing the dictionary
print(country_capitals)
# {'Germany': 'Berlin', 'Canada': 'Ottawa', 'England': 'London'}
```



- Keys of a dictionary must be immutable (Immutable objects can't be changed once created.).
- The keys of a dictionary must be unique. If there are duplicate keys, the later value of the key overwrites the previous value.

Python Dictionary

Actions

- Access Dictionary Items
  - We can access the value of a dictionary item by placing the key inside square brackets.
  - We can also use the get() method to access dictionary items.
- Add Items to a Dictionary
  - We can add an item to a dictionary by assigning a value to a new key.
- Change Dictionary Items
  - Python dictionaries are mutable (changeable). We can change the value of a dictionary element by referring to its key.
- Remove Dictionary Items
  - We can use the del statement to remove an element from a dictionary.

```
# creating a dictionary
country_capitals = {
  "Germany": "Berlin",
  "Canada": "Ottawa"
}
# Access items
den_cap = country_capitals["Germany"]
can_cap = country_capitals.get("Canada")

# add an item with "Italy" as key and "Rome"
as its value
country_capitals["Italy"] = "Rome"

# change the value of "Italy" key to "Naples"
country_capitals["Italy"] = "Naples"
country_capitals.update({"Italy": "Rome"})

# delete item having "Germany" key
del country_capitals["Germany"]

# clear the dictionary
country_capitals.clear()
```

Python Dictionary

Nested Dictionaries

- Nested dictionaries are dictionaries that are stored as values within another dictionary.
- Ex: An organizational chart with keys being different departments and values being dictionaries of employees in a given department. For storing employee information in a department, a dictionary can be used with keys being employee IDs and values being employee names. The tables below outline the structure of such nested dictionaries and how nested values can be accessed.

```
company_org_chart = {
  "Marketing": {
    "ID234": "Jane Smith"
  },
  "Sales": {
    "ID123": "Bob Johnson",
    "ID122": "David Lee"
  },
  "Engineering": {
    "ID303": "Radhika Potlapally",
    "ID321": "Maryam Samimi"
  }
}
```

Accessing nested dictionary items

```
print(company_org_chart["Sales"]["ID122"])  # -->David Lee
print(company_org_chart["Engineering"]["ID321"])  # -->Maryam Samimi
```

Python Dictionary

Python Dictionary Methods

Method	Description
dict.clear()	Remove all the elements from the dictionary
dict.copy()	Returns a copy of the dictionary
dict.get(key, default = "None")	Returns the value of specified key
dict.items()	Returns a list containing a tuple for each key value pair
dict.keys()	Returns a list containing dictionary's keys
dict.update(dict2)	Updates dictionary with specified key-value pairs
dict.values()	Returns a list of all the values of dictionary
pop()	Remove the element with specified key
popItem()	Removes the last inserted key-value pair
dict.setdefault(key,default= "None")	set the key to the default value if the key is not specified in the dictionary
dict.has_key(key)	returns true if the dictionary contains the specified key.

Python Dictionary

Exercises

1. Write python program:
   a) Convert two lists into a dictionary
   b) Merge two Python dictionaries into one
   c) Print the value of key 'history' from the below dict
   d) Initialize dictionary with default values
   e) Create a dictionary by extracting the keys from a given dictionary
   f) Delete a list of keys from a dictionary
   g) Check if a value exists in a dictionary
   h) Rename key of a dictionary
   i) Get the key of a minimum value from the following dictionary
   j) Change value of a key in a nested dictionary
2. Write a Python program that counts the number of times characters appear in a text paragraph.
3. Write a program using a dictionary containing keys starting from 1 and values containing prime numbers less than a value N.

Python Dictionary

Exercises

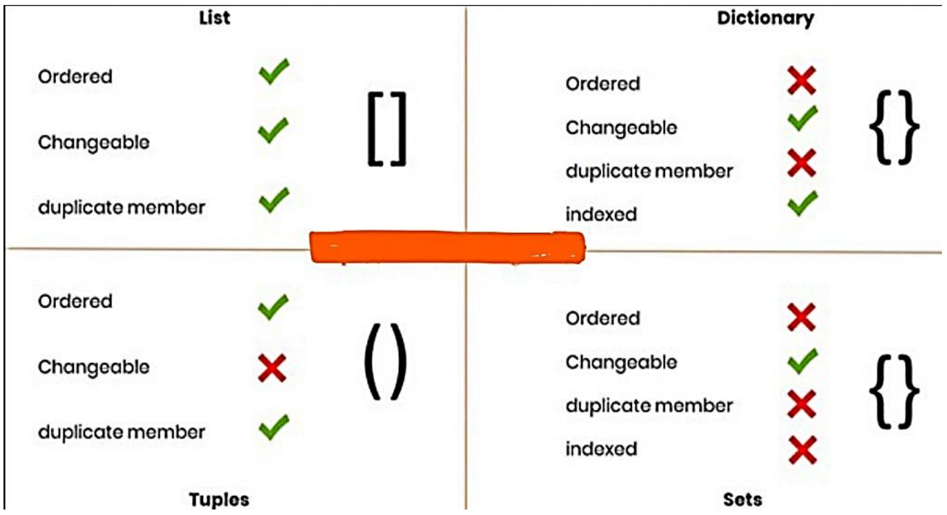
Restructuring the company data: Suppose you have a dictionary that contains information about employees at a company. Each employee is identified by an ID number, and their information includes their name, department, and salary. You want to create a nested dictionary that groups employees by department so that you can easily see the names and salaries of all employees in each department.

Write a Python program that when given a dictionary, employees, outputs a nested dictionary, dept_employees, which groups employees by department.

```
# Input:
employees = {
   1001: {"name": "Alice", "department": "Engineering",
"salary": 75000},
   1002: {"name": "Bob", "department": "Sales", "salary":
50000},
   1003: {"name": "Charlie", "department": "Engineering",
"salary": 80000},
   1004: {"name": "Dave", "department": "Marketing",
"salary": 60000},
   1005: {"name": "Eve", "department": "Sales", "salary":
55000}
}
```

```
# Resulting dictionary:
dept_employees = {
   "Engineering": {
      1001: {"name": "Alice", "salary": 75000},
      1003: {"name": "Charlie", "salary": 80000}
   },
   "Sales": {
      1002: {"name": "Bob", "salary": 50000},
      1005: {"name": "Eve", "salary": 55000}
   },
   "Marketing": {
      1004: {"name": "Dave", "salary": 60000}
   }
}
```

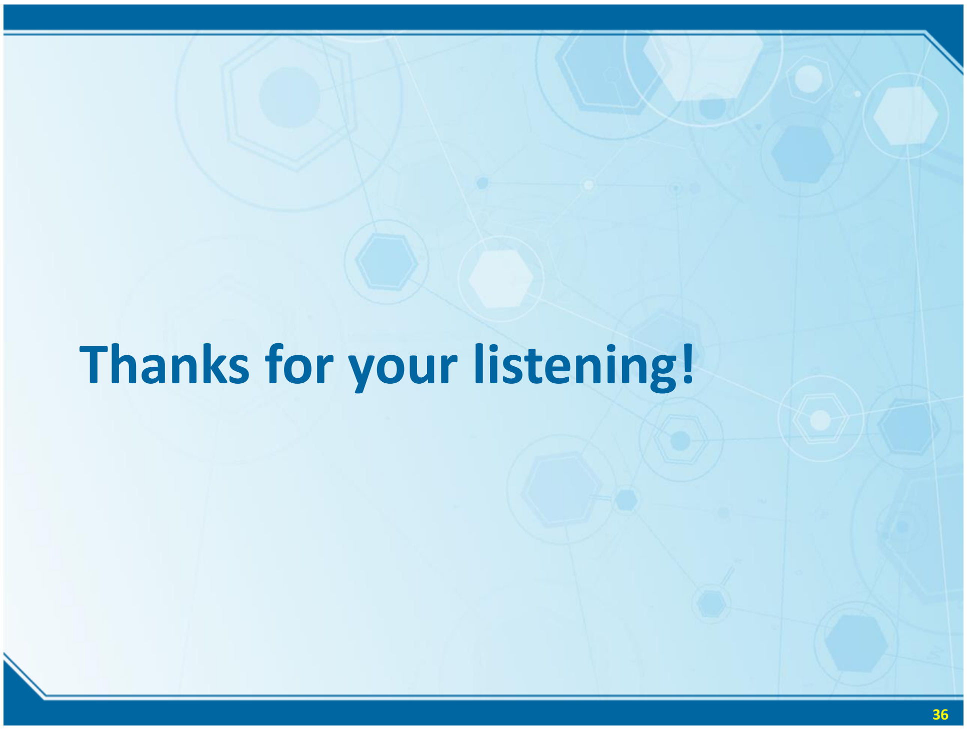
Summary



List			Dictionary		
Ordered	✓	[]	Ordered	✗	{}
Changeable	✓		Changeable	✓	
duplicate member	✓		duplicate member	✗	
			indexed	✓	

[redacted]

Tuples			Sets		
Ordered	✓	()	Ordered	✗	{}
Changeable	✗		Changeable	✓	
duplicate member	✓		duplicate member	✗	
			indexed	✗	



Thanks for your listening!

36

36