FSE+ Fondo Social Europeo Plus

Python
F.4811.015.002

Cofinanciado por la Unión Europea

GOBIERNO DE ESPAÑA · MINISTERIO DE INDUSTRIA Y TURISMO · EOI Escuela de Organización Industrial

Fondos Europeos · Sistema Nacional de Garantía Juvenil · SAMSUNG

Cofinanciado por la Unión Europea
GOBIERNO DE ESPAÑA MINISTERIO DE INDUSTRIA Y TURISMO
Escuela de Organización Industrial
Fondos Europeos
Sistema Nacional de Garantía Juvenil
SAMSUNG

SAMSUNG

Samsung Innovation Campus

Coding and Programming Course

Together for Tomorrow!
Enabling People
Education for Future Generations

Chapter 2.

Python Programming Basic - Sequence Data Type in Python

C&P Course

Chapter Description

Chapter objectives

- ✓ Learners will be able to use sequence data types such as the list, dictionary, tuple, and set of Python. Python is characterized by providing enriched data structures, including sequence data types. Learners will use those data structures to acquire advanced coding skills.

Chapter contents

- ✓ Unit 10. List and Tuple Data Types
- ✓ Unit 11. Dictionary Data Types
- ✓ Unit 12. Sequence Data Types
- ✓ Unit 13. Two-Dimensional Lists
- ✓ Unit 14. Dictionary Method 1
- ✓ Unit 15. Dictionary Method 2
- ✓ Unit 16. Set Data Types

Unit 10.

List and Tuple Data Types

Unit learning objective (1/3)

Learning objectives

- ✓ Understand the necessity and concept of list data types.
- ✓ Understand and utilize list indexing.
- ✓ Be able to calculate maximum and minimum values of the list by using various kinds of built-in functions.
- ✓ Be able to understand the differences between list and tuple and select appropriate data types for different development conditions.
- ✓ Be able to use tuple to assign values to different variables at once.
- ✓ Be able to extract necessary data through the list and tuple slicing.

Unit learning objective (2/3)

Learning overview

- ✓ Learn the differences between advanced data structures including list, dictionary, and tuple.
- ✓ Learn the characteristics of Python list data types and various methods.
- ✓ Learn the necessity and applications of list data types.
- ✓ Learn the characteristics of tuple and list data types and how to apply them.

Concepts you will need to know from previous units

- ✓ Be familiar with the functions of various Python operators such as arithmetic operators, comparison operators, logical operators and how to use them.
- ✓ How to use for() and while() loops
- ✓ How to use the input() and the split methods of strings

Unit learning objective (3/3)

Keywords

- List
- Indexing
- Slicing
- in/not in operators
- Tuple

Mission

1. Real world problem

1.1. Big data is important, but it is extremely difficult to work with it

- Today's data is often extremely massive and complex such that it is difficult to be processed with the data management software.
- A term for the large capacity data is "big data." In addition to its massive volume, big data is created very quickly, and a lot of atypical data are found.
- So, although big data is very important, it is hard to work with it.
- The big data technology used in marketing tracks the purchase history of customers to analyze when and why they purchased a certain product for strengthening commercial values of the existing product and making customer-oriented promotions.

1. Real world problem

1.2. Making a data analysis program

- The database of company A would have a lot of customer information who purchased the products of the company. Assume that the customer information includes the name, age, gender, height, weight, and many other data about the customers.
- For the convenience of saving, information of multiple customers would be in a single data value.
- When saving data in a single data value, it would be much convenient when setting orders of entering the name, address, and age of customers.
- From this mission, you will be able to make a data analysis program by entering personal information of multiple customers.

1. Real world problem

1.3. Spreadsheet?

- The first spreadsheet in the world

- Among the office programs for data processing, spreadsheet is a program that processes table type data.
- Microsoft's Excel is the most widely used spreadsheet software for office works.
- The world's first spreadsheet is the program called VisiCalc developed by VisiCorp in 1979.
- This program was designed to be performed by the computer called Apple 2.

2. Solution

2.1. Personal Database

- Assume that five different kinds of one person's information including name, age, gender, height, and weight are saved in the list.
- For instance, when a person named David Doe is a 20 years old male, is 180cm tall and weighs 100kg, his information will be expressed as a list ['David Doe', 20, 1, 180.0, 100.0].
- If a person named Jane Carter is a 22 years old female who is 169cm tall and weighs 60kg, it will be expressed as ['Jane Carter', 22, 0, 169.0, 60.0].
- The third element is gender information which will be expressed as 0 for women and 1 for men.
- When gathering information of these two people, it will be expressed as [' David Doe ', 20, 1, 180.0, 100.0, ' Jane Carter ', 22, 0, 169.0, 60.0]. Such a method to process information of multiple people is useful in real life.

2. Solution

2.2. Creating personal database

- Entering information of multiple customers and processing it all at once would be the most fundamental task in data science.
- In this mission, you will process personal information of four people. The list “person_ list” is created to integrate the information of four people as shown below. The program to calculate the average age of people in the person_list will be implemented by slicing, which is important in list. Slicing is covered in this chapter.

```
person1 = ['David Doe', 20, 1, 180.0, 100.0]
person2 = ['John Smith', 25, 1, 170.0, 70.0]
person3 = ['Jane Carter', 22, 0, 169.0, 60.0]
person4 = ['Peter Kelly', 40, 1, 150.0, 50.0]

person_list = person1 + person2 + person3 + person4
```

Results to be implemented `The average age is 26.75.`

3. Mission

3.1. How the personal database works

```python
person1 = ['David Doe', 20, 1, 180.0, 100.0]
person2 = ['John Smith', 25, 1, 170.0, 70.0]
person3 = ['Jane Carter', 22, 0, 169.0, 60.0]
person4 = ['Peter Kelly', 40, 1, 150.0, 50.0]

person_list = person1 + person2 + person3 + person4

n_persons = int (len(person_list) / 5)
age_sum = 0.0
for age in person_list[1::5]:
    age_sum += age
average_age = float(age_sum) / n_persons
print('the average age is', str(average_age))
```

```
the average age is 26.75
```

3. Mission

3.2. Programming plan

Pseudocode

[1] Start

[2] Create a list including the name, age, gender (0,1), height, and weight in the personal database.

[3] Calculate the number of persons in the personal database.

[4] Traverse the "for i in" list and skip the columns with age {

[5] Add the sum of age. }

[6] Divide the sum of age with number of persons to print average value

[7] End

Flowchart

3. Mission

3.3. Personal database final code

```python
person1 = ['David Doe', 20, 1, 180.0, 100.0]
person2 = ['John Smith', 25, 1, 170.0, 70.0]
person3 = ['Jane Carter', 22, 0, 169.0, 60.0]
person4 = ['Peter Kelly', 40, 1, 150.0, 50.0]

person_list = person1 + person2 + person3 + person4

n_persons = int (len(person_list) / 5)
age_sum = 0.0
for age in person_list[1::5]:
    age_sum += age
average_age = float(age_sum) / n_persons
print('the average age is', str(average_age))
```

Key concept

1. List and Dictionary

1.1. Efficient data processing

- The purpose of using a computer is to make data processing be efficient. Consider a case when you should save and process the test score data of many people. There would be many different variables to save test scores.
- In the code below, there are 7 number of data, so 7 different individual variables should be declared to include each value. Duplicating and operating individual elements will require many codes.

```
score0 = 87
score1 = 84
score2 = 95
score3 = 67
score4 = 88
score5 = 94
score6 = 63
print(score0, score3)
```

```
87 67
```

1. List and Dictionary

1.2. Necessity of list data types

- The memory structure consists of individual variable creation and value assignment would require many different variable names.
- Especially, adding and/or averaging many different variables could require complicated coding and occur errors.
- So, it is useful to combine them as a single data type.

1. List and Dictionary

1.3. Dictionary and necessity of list data types

- We've discussed the necessity of list data types.
- The following describes the significance of dictionary data type.
  - When entering the information provided on the right, the information below should be paired up.

```
Surname: Doe
First name: David
Work: Company A
```

  - In contrast to the list, dictionary is a data structure used to express paired information (key:value).

- Comparison with the data expression in dictionary
  - Dictionary is a data type that has a pair of key and value.
  - It is characterized by using the key to refer to the value.

2. List Declaration

2.1. What is list?

- List is a data structure that can contain an individual value in a single variable or data value.
  - It is much more convenient and efficient than writing and managing multiple variable names.
  - Duplication and operation can be done all at once.
- item or element
  - Items or elements form a list.
  - The data values are distinguished with a comma for saving.
  - Reference is importation of the list and it uses index.
  - The list declaration and element reference can be done as follows.

```
score_list = [87, 84, 95, 67, 88, 94, 63]
score_list
```

```
[87, 84, 95, 67, 88, 94, 63]
```

```
score_list = [87, 84, 95, 67, 88, 94, 63]
print(score_list[0], score_list[3])
```

```
87 67
```

The element reference uses the list name and index

2. List Declaration

2.2. List and index

- Continuous data values to save test scores of multiple people can be referred by using the score_list variable and index as follows.
- In the list, the values (or items) are distinguished by using a comma inside the [].
- It is also possible to import desired values by using an index to designate the location of the first value.

2. List Declaration

2.2. List and index

- No restrictions are applied to the data types to be included in the list.
  - Save and import the four strings including 'banana', 'apple', 'orange', 'kiwi' in the fruits list.
  - It is possible to save integers like 100, 200, 400 and string value 'apple' at the same time in the mixed_list.

```
fruits = ['banana', 'apple', 'orange', 'kiwi']  # List with strings
print(fruits)
mixed_list = [100, 200, 'apple', 400]
print(mixed_list)
```

```
['banana', 'apple', 'orange', 'kiwi']
[100, 200, 'apple', 400]
```

Python lists can have different data types including integers, real numbers, strings, etc.

2. List Declaration

2.3. The range function to create a sequence

- A list can be created by using the range or strings.
- Obtain the sequence of numbers from 1 to 9 with the function range(1, 10) and create a list containing the array of those elements by using the list function.

```
list4 = list(range(1, 10))
list4
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Use the range function of Python
to quickly create an array of continuous numbers.

3. List Indexing and Calculating Length

3.1. Definitions: List, index, indexing

Read the following definitions.

- List
  - A Python data structure that can contain multiple items and replace internal values.
  - Supports extraction of necessary values from saved items.
- index
  - Refers to the number that indicates the item value of the list.
  - The index of a list with n items will increase from 0 to n-1.
  - Using a negative index is also possible.
- indexing
  - Approaching to the data value by using the item index.

3. List Indexing and Calculating Length

3.2. List indexing and calculating length

- In the list, the index of the first item is 0, and the index of the last item is n-1.
- Use the len function to calculate the number of items in the list, which in other words the length of the list.

```
n_list = [11, 22, 33, 44, 55, 66]
print(n_list)
print(len(n_list))
```

```
[11, 22, 33, 44, 55, 66]
6
```

index	[0]	[1]	[2]	[3]	[4]	[5]
n_list =	11	22	33	44	55	66

- The length of n_list is 6, and the accessible indexing range is 0 ~ 5.

```
n_list[0]      # The index of the first item of list is 0.
```

```
11
```

```
n_list[1]      # The index of the second item of list is 1.
```

```
22
```

```
n_list[0] = 11
n_list[1] = 22
n_list[2] = 33
n_list[3] = 44
n_list[4] = 55
n_list[5] = 66
```

3. List Indexing and Calculating Length

3.3. Cautions in list indexing

Do not deviate the index range

```
n_list = [11, 22, 33, 44, 55, 66]    # Index with 6 elements
n_list[5]                            # The last element value of the list
```

```
66
```

```
n_list[6]                            # The 7th element value of the index does not exist
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-16-856e6c489946> in <module>
----> 1 n_list[6]                        # The 7th element value of the index does not exist

IndexError: list index out of range
```

- Do not insert an index value that is greater than the max. index value.
- The max. index is len(n_list)-1.
- When deviating the accessible index range, the following error occurs: IndexError: list index out of range.

3. List Indexing and Calculating Length

3.4. Negative index

- Making an approach to the element is possible by using negative indices.

```
1 n_list = [11, 22, 33, 44, 55, 66]
2 n_list[-1]
```

66

```
1 n_list[-2]
```

55

```
1 n_list[-3]
```

44

Line 2

- The last element value of the list can be approached with the negative index -1.

3. List Indexing and Calculating Length

3.4. Negative index

- The len(n_list) function returns the list's length.
- The item of the last element of the list is [-1 + len(n_list)].
- The first item is [-len(n_list) + len(n_list)]=[0]. The len(n_list) below is 6, so [-6 + 6] = [0].
- Element reference is possible by using negative indices in Python list.
- For negative indices, process indexing by reducing -1 from the last element so that it becomes -1, -2, -3.

```
n_list[-1] = 66
n_list[-2] = 55
n_list[-3] = 44
n_list[-4] = 33
n_list[-5] = 23
n_list[-6] = 11
```

n_list =

Negative index

11	22	33	44	55	66
[-6]	[-5]	[-4]	[-3]	[-2]	[-1]

4. Addition, Deletion, and Expansion of List Elements

4.1. + calculation to add lists

- Use the + calculation to add two different lists.
- In the following cases, all items of person 1 and person 2 will be included in a single list.

```
person1 = ['David Doe', 20, 1, 180.0, 100.0]
person2 = ['John Smith', 25, 1, 170.0, 70.0]

person_list = person1 + person2  # Add items of two lists to make a single list
print(person_list)
```

```
['David Doe', 20, 1, 180.0, 100.0, 'John Smith', 25, 1, 170.0, 70.0]
```

4. Addition, Deletion, and Expansion of List Elements

4.2. The append method for list element addition

- Addition or deletion of desired items is possible in the existing list.
- append method: A method to insert a new item to the end of an existing list.
- Method is the built-in function of a list and is called by using the .(dot) operator.
- Method will be discussed in detail later.

```
a_list = ['a', 'b', 'c', 'd', 'e']
a_list.append('f')   # Add 'f'
a_list
```

```
['a', 'b', 'c', 'd', 'e', 'f']
```

```
n_list = [10, 20, 30, 40]
n_list.append(50)    # Add 50
n_list
```

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

4. Addition, Deletion, and Expansion of List Elements

4.3. The extend method for list element addition

- extend method: Adds a list or item behind the list.
- As provided below, arguments [1, 2, 3] are given. If so, the elements of the [1, 2, 3] list are added to the behind of the ['a', 'b', 'c'] list which will result in ['a', 'b', 'c', 1, 2, 3].
- Inserting an individual element such as 'd' is also possible.

```
list1 = ['a', 'b', 'c']
list2 = [1, 2, 3]
list1.extend(list2)
list1
```

```
['a', 'b', 'c', 1, 2, 3]
```

```
list1.extend('d')
list1
```

```
['a', 'b', 'c', 1, 2, 3, 'd']
```

4. Addition, Deletion, and Expansion of List Elements

4.3. The extend method for list element addition

- When using the append method to the list1 below, the result [11, 22, 33, 44, 55, 66] won't be generated.
- When calling the list1.append([55, 66]) method, the list items [55, 66] will be added behind [11, 22, 33, 44], so the list1 will be [11, 22, 33, 44, [55, 66]].

```
list1 = [11, 22, 33, 44]
list1.append([55, 66])
list1
```

```
[11, 22, 33, 44, [55, 66]]
```

- When adding the items of a new list inside a list, the extend method can be used.
- Practice with making the following program.

```
list1 = [11, 22, 33, 44]
list1.extend([55, 66])
list1
```

```
[11, 22, 33, 44, 55, 66]
```

4. Addition, Deletion, and Expansion of List Elements

4.4. Methods to delete list elements

- Use the del instruction of Python
- Use the remove method in the list class
- Use the pop method
  - When using the pop method, it will delete the item in a certain position of the list and return the item at the same time.

4. Addition, Deletion, and Expansion of List Elements

4.5. Deletion with the del keyword

- Delete the item in the designated index
- Caution: Deletion is not possible with the del 44 method.

```
1 n_list = [11, 22, 33, 44, 55, 66]
2 print(n_list)  # Print the entire items
3
4 del n_list[3]  # Delete 44
5 print(n_list)
```

```
[11, 22, 33, 44, 55, 66]
[11, 22, 33, 55, 66]
```

Line 4

- Use the index as the n_list[3] to delete the fourth element 44.

4. Addition, Deletion, and Expansion of List Elements

4.6. Deletion with the pop method

- The pop method returns and delete the last item of the list.

```
n_list = [10, 20, 30]
print(n_list) # print the entire items
```

```
[10, 20, 30]
```

```
n = n_list.pop()
print('n =', n)
print('n_list =', n_list)
```

```
n = 30
n_list = [10, 20]
```

`n_list = [10, 20, 30] :`

`after n = n_list.pop:`

4. Addition, Deletion, and Expansion of List Elements

4.7. Deletion with the remove method

- Delete a certain value from the list by using the list's method.
- Use the same method as remove(44).

```
1 n_list = [11, 22, 33, 44, 55, 66]
2 print(n_list)
3
4 n_list.remove(44)
5 print(n_list)
```

```
[11, 22, 33, 44, 55, 66]
[11, 22, 33, 55, 66]
```

Line 4

- Use the remove method of n_list to delete the item with the vale 44.

4. Addition, Deletion, and Expansion of List Elements

4.8. Cautions when deleting with the remove method

Problems of the remove method

```python
n_list = [11, 22, 33, 44, 55, 66]
print(n_list)

n_list.remove(88)
print(n_list)
```

```
[11, 22, 33, 44, 55, 66]

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-46-e5b8ba8b9713> in <module>
      2 print(n_list)
      3
----> 4 n_list.remove(88)
      5 print(n_list)

ValueError: list.remove(x): x not in list
```

- An error occurs when deleting an invalid item with the remove method.

5. List Slicing

5.1. Slicing

- Slicing: refers to slicing of specific sections of a list
- Use the list_name [start : end] syntax to state the section. Slicing is possible until end-1 index.

Result of the a_list[1:5] slicing: Sliced until 50, which is the 5-1 index

5. List Slicing

5.2. Slicing example

List slicing is possible as follows in Python.

```
1 a_list = [10, 20, 30, 40, 50, 60, 70, 80]
2 a_list[1:5]
```

```
[20, 30, 40, 50]
```

```
a_list[0:5]
```

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

```
a_list[1:]
```

```
[20, 30, 40, 50, 60, 70, 80]
```

```
a_list[:5]
```

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

```
a_list[:]
```

```
[10, 20, 30, 40, 50, 60, 70, 80]
```

Line 2

- Sliced from the second element 20 to the fifth element 50
- The result is 20, 30, 40, 50.

5. List Slicing

5.2. Slicing example

```
1 a_list = [10, 20, 30, 40, 50, 60, 70, 80]
2 a_list[1:5]
```

```
[20, 30, 40, 50]
```

```
1 a_list[0:5]
```

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

```
1 a_list[1:]
```

```
[20, 30, 40, 50, 60, 70, 80]
```

```
1 a_list[:5]
```

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

```
1 a_list[:]
```

```
[10, 20, 30, 40, 50, 60, 70, 80]
```

Line 1

- Sliced from the first element 10 to the fifth element 50
- The result is 10, 20, 30, 40, 50.

5. List Slicing

5.2. Slicing example

```
1 a_list = [10, 20, 30, 40, 50, 60, 70, 80]
2 a_list[1:5]
```

```
[20, 30, 40, 50]
```

```
1 a_list[0:5]
```

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

```
1 a_list[1:]
```

```
[20, 30, 40, 50, 60, 70, 80]
```

```
1 a_list[:5]
```

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

```
1 a_list[:]
```

```
[10, 20, 30, 40, 50, 60, 70, 80]
```

Line 1

- Sliced from the second element 20 to last element.
- If slicing until the last element, you can skip the second index.

5. List Slicing

5.2. Slicing example

```
1 a_list = [10, 20, 30, 40, 50, 60, 70, 80]
2 a_list[1:5]
```

```
[20, 30, 40, 50]
```

```
1 a_list[0:5]
```

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

```
1 a_list[1:]
```

```
[20, 30, 40, 50, 60, 70, 80]
```

```
1 a_list[:5]
```

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

```
1 a_list[:]
```

```
[10, 20, 30, 40, 50, 60, 70, 80]
```

Line 1

- Sliced from the first element 10 to the fifth element 50.
- You can skip the index of the first element.

5. List Slicing

5.2. Slicing example

```
1 a_list = [10, 20, 30, 40, 50, 60, 70, 80]
2 a_list[1:5]
```

```
[20, 30, 40, 50]
```

```
1 a_list[0:5]
```

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

```
1 a_list[1:]
```

```
[20, 30, 40, 50, 60, 70, 80]
```

```
1 a_list[:5]
```

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

```
1 a_list[:]
```

```
[10, 20, 30, 40, 50, 60, 70, 80]
```

Line 1

- When omitting all indices, slice from the first element to the last element.
- You can also slice the entire list with [:].

5. List Slicing

5.3. Summary of list slicing

Python slicing supports negative indices and negative slice step values.

Syntax	Function
a_list[start:end]	Slicing the items from the start to (end-1) (does not include the items of the end index)
a_list[start:]	Slicing from the start to the end of the list. Slicing the end part of the list.
a_list[:end]	Slicing from the start to the end-1 index
a_list[:]	Slicing the entire list
a_list[start:end:step]	Slicing by skipping a step from start to end-1
a_list[-2:]	Slicing two items from the end
a_list[-2:]	Slicing entire items from the start except for the last two items
a_list[::-1]	Import all the items but slicing in reverse order
a_list[1::-1]	Slicing the first two items in reverse order

5. List Slicing

5.4. List slicing with the start and end indices stated

- a_list[1:5]: Import the items from a_list[1] to a_list[5-1].

```
a_list = [10, 20, 30, 40, 50, 60, 70, 80]
a_list[1:5]
```

```
[20, 30, 40, 50]
```

5. List Slicing

5.5. Omitting the last index

- a_list[1:]: Import until the last item of the list

```
a_list[1:]
```

```
[20, 30, 40, 50, 60, 70, 80]
```

5. List Slicing

5.6. Omitting both the start and end indices

- [:] method slicing: Import all items of the list

```
a_list[:]
```

```
[10, 20, 30, 40, 50, 60, 70, 80]
```

5. List Slicing

5.7. List slicing of negative indices

- The index of the end element becomes -1, and the elements in front of it are assigned -2, -3, ...

a_list	10	20	30	40	50	60	70	80
Negative index	-8	-7	-6	-5	-4	-3	-2	-1

5. List Slicing

5.8. Slicing with using negative indices

- a_list[-7:-2]: Import the items from a_list[-7] to a_list[-2-1] by using negative index range. As a result, it will include [20, 30, 40, 50, 60].

```
a_list[-7:-2]
```

```
[20, 30, 40, 50, 60]
```

5. List Slicing

5.9. Omitting the last index from negative indices

If omitting the last index when using negative indices, it will import until the last item of the list as follows.

```
a_list[-7:]
```

```
[20, 30, 40, 50, 60, 70, 80]
```

5. List Slicing

5.10. Omitting the first index from negative indices

- Omitting the first index is possible for slicing using negative indices.
- a_list[:-2]: Import the indices from the first item to (-2-1) = -3 index.

```
a_list[:-2]
```

```
[10, 20, 30, 40, 50, 60]
```

6. Examining Internal List Values

6.1. in operator

- "in" operator returns True or False. After examining to see if an element is at a specific sequence, if it is found, it returns True, and False if not. (The "not in" operator returns the opposite.)
- However, the for() can list each of the item by using the "in" operator.
- It is used to examine if a specific element is included in the data structures such as strings, lists, and tuple.

```
a_list = [10, 20, 30, 40]
10 in a_list
```

```
True
```

Before proceeding with the a_list.remove(10) method, check if the item 10 exists with the "in" operation and if it returns True, and then making a deletion won't result an error.

```
50 in a_list
```

```
False
```

```
10 not in a_list
```

```
False
```

```
50 not in a_list
```

```
True
```

```
for n in a_list:
    print(n, end=' ')
```

```
10 20 30 40
```

6. Examining Internal List Values

6.2. Using the in operator

- Use the in operator of Python.
- If the element is included in the list, then the in operator returns True, and if not, it returns False.

```
n_list = [11, 22, 33, 44, 55, 66]

print(88 in n_list)        # 88 is not in the n_list
print(55 in n_list)        # 55 is in the n_list
```

```
False
True
```

6. Examining Internal List Values

6.2. Using the in operator

- Prevent an error in advance by using “in” operation.
- Check if there’s a value that will be deleted. Delete with the remove method only if there’s a value for deletion.

```
n_list = [11, 22, 33, 44, 55, 66]
if (55 in n_list) :      # If 55 is an element of the list
    n_list.remove(55)    # Delete 55 from the list
if (88 in n_list) :      # If 88 is an element of the list
    n_list.remove(88)    # Delete 88 from the list
print(n_list)
```

```
[11, 22, 33, 44, 66]
```

7. Tuple

7.1. Immutable and mutable data types

- Python data types are classified in immutable and mutable data types as provided in the figure below. Tuple is extremely similar with the list. However, the data of tuple cannot be changed once designated, and it is a representative immutable data type.
- Once designated, tuple data cannot be changed, so its simple structure and quick access speed compared to the list are advantages of tuple. Choose an appropriate data type depending on the purpose.

7. Tuple

7.2. Data type in which the values cannot be changed once created: Tuple

- Create a simple tuple as follows.
- It seems similar with the list, but there are differences between them.

```
colors = ("red", "green", "blue")
colors
```

```
('red', 'green', 'blue')
```

Tuple for saving colors

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

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

Tuple for saving multiple integers

7. Tuple

7.2. Data type in which the values cannot be changed once created: Tuple

Tuple is an immutable object.

```
t1 = (1, 2, 3, 4, 5)
t1[0] = 100
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-107-614bcaadef71> in <module>
      1 t1 = (1, 2, 3, 4, 5)
----> 2 t1[0] = 100

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

- The most important characteristic of the tuple is that the values won't be changed.
- If trying to change the internal values of tuple as shown in the code above, TypeError occurs.
- An error also occurs when trying to delete the element value by using del t1[0].

Paper coding

Try to fully understand the basic concept before moving on to the next step.

Lack of understanding basic concepts will increase your burden in learning this course, which may make you fail the course.

It may be difficult now, but for successful completion of this course we suggest you to fully understand the concept and move on to the next step.

Paper coding

Q1. Create prime_list that has prime numbers between 2~10 as its elements. Then, use list indexing to the first element of the list and print as shown below.

Conditions for Execution	`1st element of prime_list: 2`
Time	5 min

Write the entire code and the expected output results in the note.

Paper coding

Q2. Create prime_list that has prime numbers between 1~10 as its elements. Then, use the append method to add 11. Print the results before and after addition as shown below.

Conditions for Execution	`Prime numbers : [2,3,5,7]` `Prime numbers after addition : [2,3,5,7,11]`
Time	5 min

Write the entire code and the expected output results in the note.

Paper coding

Q3. For the list1 and list2, use the nested for loop to multiply each element of list1 and list2 and then print the result with the element multiplication result.

Conditions for Execution	Declare list1 and list2 in the first and second rows. Use the nested for loop in the third and fourth row, and use the print loop in the fifth row.
Time	5 min

```
list1= = [3,5,7]
list2 = [2,3,4,5,6]
```

Output example

```
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
```

Write the entire code and the expected output results in the note.

Unit 10. List and Tuple Data Types

Let's code

1. What is the list and why is it important?

1.1. Significance of the list

Try to save the height of different people. When programming it, it will be as follows.

```
height1 = 178.9   # Save the float type data.
height2 = 173.5   # Save the float type data.
height3 = 166.1   # Save the float type data.
height4 = 164.3   # Save the float type data.
height5 = 176.4   # Save the float type data.
```

" If there are 100 people, there are 100 different variables.
So, if there are 1,000 people, then the variables would be...??? "

1. What is the list and why is it important?

1.1. Significance of the list

If so, create a list and save data all at once. In Python, you can create a list with []. For instance, write as following to save the height of 5 different students in a list.

```
heights = [178.9, 173.5, 166.1, 164.3, 176.4]
heights
```

```
[178.9, 173.5, 166.1, 164.3, 176.4]
```

Use a list to contain multiple values in a single variable.

Data expressed in individual variables

Data expressed in the list

1. What is the list and why is it important?

1.2. Create a list with multiple items

- In Python, the list can be created as other variables. Write the items in [] and save as variables, then the list variable will be created. The following is an example of a simple list made with 3 members of BTS.

```
bts = ['V', 'Jungkook', 'Jimin']
```

- If there are no members in the beginning, it can be expressed as follows. If so, an empty list is created.

```
bts = []
```

1. What is the list and why is it important?

1.2. Create a list with multiple items

- How can an empty list without any items be used? Items can be added to the empty list by using codes. Most of the time, we cannot predict how many items will be included in the list, and if so, using an empty list is helpful. Use the append method to newly add items to the list.

```
bts = []
bts.append("V")
bts
```

```
['V']
```

- The most convenient way to add an element in a list is to use the + operator.

```
bts = []
bts = bts + ["V"]
bts
```

```
['V']
```

2. Calculating the length, max. value, and sum of the list

2.1. Built-in functions used in the list

- If the following values are found in the list, built-in functions including len, max, min, sum, any can be used for convenience.
  - len returns the length of the list
  - max returns the maximum item value of the list, and min returns the minimum item value of the list
  - If the list has numerical items, sum adds up all of the elements.

```
n_list = [200, 700, 500, 300, 400]
len(n_list)
```

```
5
```

```
max(n_list)
```

```
700
```

```
min(n_list)
```

```
200
```

```
sum(n_list)
```

```
2100
```

2. Calculating the length, max. value, and sum of the list

2.2. Applicable functions for the list

- It is possible to add the value created from the range function to the list through list(range()) as shown below.
- Among the applicable functions for the list, the any function returns True if there is at least one element that is not 0. any(n_list) returns True.
- In the a_list consists of 0 and '' (whitespace), the any function returns False.

```
list(range(1,11))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
n_list = [200, 700, 500, 300, 400]
a_list = [0, '']
```

```
any(n_list)
```

```
True
```

```
any(a_list)
```

```
False
```

2. Calculating the length, max. value, and sum of the list

2.3. Approaching to the list item using index

- How would you do to take saved data from Python list? Assume that there's a list saving alphabets as following.

```
letters = ['A', 'B', 'C', 'D', 'E', 'F']
```

- To extract data from the list, use integer as index. The first data of the list will have index 0, and the second element will have index 1. Other elements will be assigned with indices in a similar way.

```
letters[0]  # Approaching to the first item of the list
```

```
'A'
```

letters	'A'	'B'	'C'	'D'	'E'	'F'
Index	0	1	2	3	4	5

2. Calculating the length, max. value, and sum of the list

2.3. Approaching to the list item using index

```
letters[1]
```

```
'B'
```

```
letters[2]
```

```
'C'
```

```
letters[-1]
```

```
'F'
```

In Python, negative indices can be used. It is useful to obtain the data that is at the end of the list.

letters	'A'	'B'	'C'	'D'	'E'	'F'
Index	0	1	2	3	4	5
Negative index	-6	-5	-4	-3	-2	-1

3. Adding list elements

3.1. Operating the list element value

Operate the element values of the list

- To add an item by using a function, use append or insert functions. While the append function will add an item at the end of the list, insert(index, item) adds an item to a designated index.

```
slist = ['David', 178.9, 'John', 173.5, 'Jane', 176.1]
print(slist)
slist.insert(4, "Petter")
slist.insert(5, 168.1)
print(slist)
```

```
['David', 178.9, 'John', 173.5, 'Jane', 176.1]
['David', 178.9, 'John', 173.5, 'Petter', 168.1, 'Jane', 176.1]
```

4. Deleting list elements

4.1. How to delete list item

- Previously, we've learned operators and methods to add or change list items. It is also possible to delete list items. For deletion, use the remove and pop methods and del command.
- Before using the remove function, first investigate if the item is included in the list with the "in" operator.

```
bts = [ "V", "J-Hope", "Suga", "Jungkook" ]
bts.remove("Jungkook")
bts
```

remove method deletes a designated item from the list

```
['V', 'J-Hope', 'Suga']
```

```
if 'Suga' in bts:
    bts.remove('Suga')
print(bts)
```

"in" investigates if an item is included in the list or tuple. It becomes a safe program when deleting this item from the BTS list only if the condition is True.

```
['V', 'J-Hope']
```

- The remove method deletes a designated item from the list as provided.

4. Deleting list elements

4.1. How to delete list item

- There are different ways to delete an item.
  - The pop also deletes an item from the list. Unlike the remove, it deletes the last item of the list and returns the item value.

```
bts = ["V", "J-Hope", "Suga", "Jungkook"]
last_member = bts.pop()  # Delete and return the last item 'Jungkook'
print(last_member)
print(bts)
```

```
Jungkook
['V', 'J-Hope', 'Suga']
```

4. Deleting list elements

4.2. Cautions for deleting list item

del command is not a list method.

- The del command is not a list method. This command is a Python keyword, and it can delete a specific item from the memory by using index as shown below.
- When using the del command as a method, SyntaxError occurs.

```
bts = [ "V", "J-Hope", "Suga", "Jungkook"]
del bts[0]    # Command to delete the first item of the list
bts
```

```
['J-Hope', 'Suga', 'Jungkook']
```

```
bts[0].del
```

```
  File "<ipython-input-5-390ae27b77c9>", line 1
    bts[0].del
           ^
SyntaxError: invalid syntax
```

5. List slicing

5.1. Slicing method

- Slicing cuts the list in a desired form
  - To make a new list by selecting some elements from a list, use slicing. For slicing a list, use a colon as following to designate a range.

```
letters = ['A', 'B', 'C', 'D', 'E', 'F']
```

```
letters[2:5]  # Slicing from the 3rd item to 5th item of the list
```

```
['C', 'D', 'E']
```

letters	'A'	'B'	'C'	'D'	'E'	'F'
Index	0	1	2	3	4	5
			'C'	'D'	'E'	
letters[2:5]			2	3	4	

5. List slicing

5.1. Slicing method

- Slicing a list does not damage the original list. It creates and returns a new list. In other words, the slice is a partial copy of the original list.
- If the first index is omitted, slicing is done from the start. The following code will do slicing from the 1st item to the 3rd item.

```
letters[:3]
```

```
['A', 'B', 'C']
```

- If the second index is omitted, slicing is done to the end of the list, which is a very convenient function.

```
letters[3:]
```

```
['D', 'E', 'F']
```

5. List slicing

5.1. Slicing method

- Slicing a list does not damage the original list. It creates and returns a new list. In other words, the slice is a partial copy of the original list.

```
letters[:]
```

```
['A', 'B', 'C', 'D', 'E', 'F']
```

The colon refers to from the start to the end

```
letters[::2]
```

```
['A', 'C', 'E']
```

Read the list from the start to the end, but skip with the step

```
letters[::-1]      # Read from the back to front to create elements
```

```
['F', 'E', 'D', 'C', 'B', 'A']
```

6. Different methods of the list

6.1. Sorting methods

- Sorting method - sort
  - Ascending order sorting is default
  - Writes a dot (.) and “sort” after the list variable

- Ascending order sorting
  - Sorting with increased value

- Descending order sorting
  - Sorting with decreased value

6. Different methods of the list

6.2. Sorting methods: sort

- Use the sort method of the list for ascending order sorting of the list elements. If adding reverse = True keyword argument, it becomes descending order sorting.
  - Keyword argument refers to an argument that commands a method or function, and it will be discussed later.

```
list1 = [20, 10, 40, 50, 30]
list1.sort()                    # Ascending order sorting of the list1 elements
list1
```

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

```
list1.sort(reverse = True)
print(list1)
```

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

6. Different methods of the list

6.3. Methods provided by the list

Method	Function
index(x)	Finds a place by using the element x.
append(x)	Adds the element x to the end of the list.
count(x)	Returns the number of object elements within the list.
extend([x1, x2])	Inserts the [x1, x2] list to the list.
insert(index, x)	Adds the x to the desired index location.
remove(x)	Deletes the element x from the list.
pop(index)	Deletes the element on the index location and then returns it. If so, the index can be omitted, which will delete and return the last element of the list.
sort()	Sorts the values in the ascending order. If the reverse argument value is True, descending order sorting is done.
reverse()	Makes a reverse order of the elements in the list.

Pair programming

Pair Programming Practice

Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.

Pair Programming Practice

Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

Collaborative learning environment

A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor's question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students "divide and conquer." When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

Pair programming

Q1. There is a list with the string s_list = ['abc', 'bcd', 'bcdefg', 'abba', 'cddc', 'opq']. Implement the following function to this list.

- Do not use the min function or sort method to print the shortest string from the strings of s_list. (If there are multiple shortest strings, print the string that shows the first as following.)

Output example

```
The shortest string : abc
```

Pair programming

Q2. There is a list with the string s_list = ['abc', 'bcd', 'bcdefg', 'abba', 'cddc', 'opq']. Implement the following function to this list.

- Do not use the min function or sort method to print the shortest string from the strings of s_list. (If there are multiple shortest strings, print the string that shows the first as following.)

Output example

```
The longest string : bcdefg
```

Pair programming

Q3. There is a list with the string s_list = ['abc', 'bcd', 'bcdefg', 'abba', 'cddc', 'opq']. Implement the following function to this list.

From the pair programming problem earlier, the length of 'abc', 'bcd', 'opq' are the same as 3. Likewise, if the string lengths are the same, write a program that prints all of the three shortest strings as follows. Use the sort(key=len) function to sort the strings by length and then write a code.

Output example

```
The shortest strings : 'abc', 'bcd', 'opq'
```

Unit 11.

Dictionary Data Types

Unit learning objective (1/3)

Learning objectives

- ✓ Be able to create dictionary data type objects and items with data values.
- ✓ Understand the differences between the dictionary and list and be able to apply appropriate data types for different conditions.
- ✓ Be able to approach to the dictionary key and assign values by using the key.

Unit learning objective (2/3)

Learning overview

- ✓ Learn how to use dictionary data types to extract values using a key.
- ✓ Learn how to approach to the dictionary key and assign values by using the key.
- ✓ Learn how to return the value of dictionary object by using the pop method provided by dictionary.
- ✓ Learn the json data type and how to convert the data to dictionary.

Concepts you will need to know from previous units

- ✓ Knowledge about the list and tuple to process multiple data as a single data
- ✓ How to approach the items of the list and tuple by using indices
- ✓ How to do slicing to cut a part of the sequence

Unit learning objective (3/3)

Keywords

- Dictionary
- Key-Value
- Key approach and assignment
- json

Mission

1. Real world problem

1.1. Kiosk order is not friendly for older people.

- Due to the worldwide spread of pandemic, many things have now become contactless.
- The spread of COVID-19 resulted in governmental support in Korea to install unmanned kiosks in cafés and restaurants.
- A survey showed that people in their 20s and 30s are highly satisfied with making orders with a kiosk.
- However, there are still many people including elderly and others who are not familiar with using digital technology that find using non-contact services difficult.
- So, we will be learning about basic technology to make good services for the customers through the mission to introduce and print café menus.

1. Real world problem

1.2. How to digitally combine two data types as one

- Most cafés show the beverage and its price together, as a set.
- So, if combining ”beverage:price” as a single bundle, it would be easy to write an order program for unmanned café.
- However, when listing the beverage and price by using the list, it would be difficult to work with because the they have different data types.
- The problem is to find a method to combine the menu and price together at all times by integrating two different data types as one.

1. Real world problem

1.3. Making a café menu program

- In this mission, a data structure called dictionary will be used to print the menu and price of a café.
- This café has five menus including Americano, Iced Americano, Cappuccino, Caffe Latte, and Espresso, and the prices are 3,000 KRW, 3,500 KRW, 4,000 KRW, 4,500 KRW, and 3,600 KRW, respectively.
- Also, try to distinguish if the item is in the menu or not for printing when the user enters an item.

2. Mission

2.1. How the café menu works #1

```
menu = {'Americano': 3000, 'Ice Americano': 3500, 'Cappuccino' : 4000, 'Cafe Latte':4500, 'Espresso':3600 }
for key in menu:
    print("{:16s} price : {:,}KRW".format(key, menu[key]))
```

```
Americano        Price : 3,000 KRW
Ice Americano    Price : 3,500 KRW
Cappuccino       Price : 4,000 KRW
Cafe Latte       Price : 4,500 KRW
Espresso         Price : 3,600 KRW
```

2. Mission

2.1. How the café menu works #2

```python
print('Americano       Price: 3,000 KRW')
print('Ice Americano   Price: 3,500 KRW')
print('Cappuccino      Price: 4,000 KRW')
print('Cafe Latte      Price: 4,500 KRW')
print('Espresso        Price: 3,600 KRW')
choice = input('Select one menu from the list: ')
if choice in menu.keys ():
    print('{} is {:,} KRW. Please make a payment.'.format(choice, menu[choice]))
else:
    print('Sorry. {} is not in the menu.'.format(choice))
```

```
Americano       Price: 3,000 KRW
Ice Americano   Price: 3,500 KRW
Cappuccino      Price: 4,000 KRW
Cafe Latte      Price: 4,500 KRW
Espresso        Price: 3,600 KRW
Select one menu from the list: Cafe Latte
Cafe Latte is 4,500 IKRW. Please make a payment.
```

2. Mission

2.2. Programming plan #1

Pseudocode

[1] Start

[2] Enter the menu and price

[3] for key in the dictionary, as the number of key for printing do {

[4] print the menu. }

[5] End

Flowchart

2. Mission

2.2. Programming plan #2

Pseudocode

[1] Start

[2] Print the menu of choice

[3] Select one menu to eat

[4] if the drink I chose is in the menu, then{

[5] print the menu }

[6] else{ it's not in the menu. print }

[7] End

Flowchart

2. Mission

2.3. Café menu final code #1

```
menu = {'Americano': 3000, 'Ice Americano': 3500, 'Cappuccino': 4000, 'Cafe Latte':4500, 'Espresso':3600 }
for key in menu:
    print("{:16s} price : {:,}KRW".format(key, menu[key]))
```

2. Mission

2.3. Café menu final code #2

```
print('Americano       Price: 3,000 KRW')
print('Ice Americano   Price: 3,500 KRW')
print('Cappuccino      Price: 4,000 KRW')
print('Cafe Latte      Price: 4,500 KRW')
print('Espresso        Price: 3,600 KRW')
choice = input('Select one menu from the list: ')
if choice in menu.keys ():
    print('{} is {:,} KRW. Please make a payment.'.format(choice, menu[choice]))
else:
    print('Sorry. {} is not in the menu.'.format(choice))
```

Key concept

1. Dictionary Declaration

1.1. Definition of Dictionary

- In this unit, we'll discuss dictionary, which is an extremely useful data structure of python.
- Dictionary is a data type that has key-value pair.
- It is a widely used data structure since it refers to the value by using the key.
- Dictionary is very similar with a list, but its unique characteristic is that it has a key-value pair.
- Also, since the dictionary looks up the value by using a key, the keys cannot be duplicated.

1. Dictionary Declaration

1.1. Definition of Dictionary

- Dictionary item is defined by being connected with a colon as [key] : [value].
- Dictionary consists of more than one item.
- The [value] can be looked up by using the [key], but the [value] cannot be used to look up the [key].

1. Dictionary Declaration

1.2. Comparison of the list and dictionary

- In the list, indices can be used to approach each value [value 1], [value 2], ... [value n]. Each value is the item.
- Indices of the list is from 0 to n-1.
- On the other hand, the key-value pair is one item in the dictionary.

1. Dictionary Declaration

1.3. Dictionary syntax

- This is the syntax that defines a list.

```
List name = [[value 1], …]
```

- The following is an example to create the fruits list.

```
fruits = ['banana', 'apple', 'orange', 'kiwi']  # List with string
```

- This is the syntax that defines a dictionary. The key-value pair is combined with a colon.

```
Dictionary name = {[height 1] : [value 1], …}
```

```
person = {'Name' : 'David Doe', 'Age' : 26, 'Weight' : 82 } # Dictionary with name, age, weight
```

2. Key-Value Pair of Dictionary

2.1. Value printing by using the dictionary key

Create a dictionary with the following information. As shown in the code below, it is possible to look up 'David Doe' through the person dictionary key 'Name,' and the same method applies to 'Age' and 'Weight.'

Item	key	value
Item 1	'Name'	'David Doe'
Item 2	'Age'	26
Item 3	'Weight'	82

```
person = {'Name': 'David Doe', 'Age': 26, 'Weight': 82}
```

```
person['Name']
```

```
'David Doe'
```

```
person['Age']
```

```
26
```

```
person['Weight']
```

```
82
```

2. Key-Value Pair of Dictionary

2.2. Example code for dictionary key-value

Creates a new dictionary named “students” that have [ID number] : [Name] pairs of three students.

Item	key	value
Item 1	2019001	‘John Smith
Item 2	2019002	‘Jane Carter’
Item 3	2019003	‘Peter Kelly’

```
students = { 2019001:'John Smith', 2019002:'Jane Carter', 2019003:'Peter Kelly'}
students[2019001]
```

```
'John Smith'
```

```
students[2019002]
```

```
'Jane Carter'
```

```
students[2019003]
```

```
'Peter Kelly'
```

3. Key Approach and Value Assignment of Dictionary

3.1. Inserting a new item into dictionary

- Use the following method to insert a new item into dictionary.

```
Dictionary name [key] = value
```

- For deletion, enter the key of the dictionary item to be deleted next to the del keyword.
- KeyError occurs when trying to delete an item with invalid key.

3. Key Approach and Value Assignment of Dictionary

3.2. Item insertion and change in dictionary

- Example of inserting an item into the dictionary: Create an item with the key 'Job'

```
person = {'Name' : 'David Doe', 'Age' : 26, 'Weight' : 82 }
```

```
person['Job'] = 'Data Scientist'  # New key:insert the item of the value
```

```
person
```

```
{'Name': 'David Doe', 'Age': 26, 'Weight': 82, 'Job': 'Data Scientist'}
```

- Modifying the dictionary item: Modify the item value from 26 to 27 with the key 'Age'

```
person = {'Name' : 'David Doe', 'Age' : 26, 'Weight' : 82 }
```

```
person['Age'] = 27
```

```
person
```

```
{'Name': 'David Doe', 'Age': 27, 'Weight': 82}
```

3. Key Approach and Value Assignment of Dictionary

3.3. Deleting an item in dictionary

- Deleting the dictionary item: Delete 'Age': 26 with the del command

```
person = {'Name' : 'David Doe', 'Age' : 26, 'Weight' : 82 }
```

```
del person['Age']
```

```
person
```

```
{'Name': 'David Doe', 'Weight': 82}
```

3. Key Approach and Value Assignment of Dictionary

3.3. Deleting an item in dictionary

Cautions when deleting a dictionary item – KeyError occurs when deleting with an invalid key

```
person = {'Name' : 'David Doe', 'Age' : 26, 'Weight' : 82 }
```

```
del person ['Hometown']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[28], line 1
----> 1 del person ['Hometown']

KeyError: 'Hometown'
```

- As shown above, KeyError occurs when trying to delete an item by using the invalid key 'Hometown.'

3. Key Approach and Value Assignment of Dictionary

3.4. Functions and operators for dictionary

- The applicable functions and operators for the dictionary are similar to those of the list.
- len function
  - Used to obtain the number of items in the dictionary
- in operator
  - Returns True if the key is present in the dictionary
- not in operator
  - Returns True if the key is not found in the dictionary
- ==, != operator
  - Checks if two dictionaries have the same item

3. Key Approach and Value Assignment of Dictionary

3.4. Functions and operators for dictionary

- The following shows how to look up dictionary items by using the “in” operator. The “not in” operator returns an opposite result from the “in” operator.
- The key ‘Name’ is found in the person dictionary key, so it returns True, while ‘Job’ returns False.

```
person = {'Name' : 'David Doe', 'Age' : 26, 'Weight' : 82 }
```

```
len(person) # Returns the number of items in dictionary
```

```
3
```

```
'Name' in person # 'Name' is found in the key
```

```
True
```

```
'Job' in person # 'Job' is not found in the key
```

```
False
```

```
'David Doe' in person # 'David Doe' is found in the value but not in the key (caution)
```

```
False
```

```
'David Doe' not in person # Returns True because 'David Doe' is not found in the key
```

```
True
```

3. Key Approach and Value Assignment of Dictionary

3.4. Functions and operators for dictionary

- Remember that the value, not the key, does not affect the “in” operator.
- “David Doe’ is included in the value, but because it’s not found in the key, the “in” operator returns False.

```
'David Doe' in person # 'David Doe' is found in the value but not in the key (caution)
```

```
False
```

```
'David Doe' not in person # Returns True because 'David Doe' is not found in the key
```

```
True
```

3. Key Approach and Value Assignment of Dictionary

3.4. Functions and operators for dictionary

- == operator returns True if two dictionaries have the same item, and False if not.
- != operator returns the opposite value.

```
d1 = {'Name' : 'David Doe', 'Age' : 26 }
```

```
d2 = {'Age' : 26, 'Name' : 'David Doe' }
```

```
d1 == d2
```

```
True
```

```
d1 != d2
```

```
False
```

3. Key Approach and Value Assignment of Dictionary

3.5. Comparison operation is not applicable in dictionary

Comparison operation in dictionary – TypeError

```
d1 > d2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[47], line 1
----> 1 d1 > d2

TypeError: '>' not supported between instances of 'dict' and 'dict'
```

- Dictionary does not support comparison operators such as >, >=, <, <=.

4. json Data Dictionary / List Conversion

4.1. Web service standard document

- The most commonly used standard document for web services is HTML (HyperText Markup Language) document.
- The HTML document is a subset of XML document.
- XML (eXtensible Markup Language) is one type of file that expresses structural data through data text document.

Web service | Web browser | HTML document

4. json Data Dictionary / List Conversion

4.2. json type

- Consider a method to deliver significant data object by using a text document.
- What would be effective methods to deliver the information such as “Name“:”David Doe”?
- A representative text form is JSON.
  - JSON is a text format that can quickly read and write the data than XML.
  - In this unit, we’ll briefly learn how to use JSON.

personal.json

```
1  {
2    "personnel": {
3      "person": [
4        {
5          "id": "harris.anderson",
6          "photo": "personal-images/harris.an
7          "name": {
8            "given": "Harris",
9            "family": "Anderson"
10         },
11         "email": "harris.anderson@example.c
12         "link": {"subordinates": "robert.ta
13         "url": {"href": "http://www.example
14       },
15       {
16         "id": "robert.taylor",
17         "photo": "personal-images/robert.ta
18         "name": {
19           "given": "Robert",
20           "family": "Taylor"
```

personal.xml

personnel person name given

```
1  <?xml version="1.0" encoding="UTF-8"?
2  <personnel>
3      <person id="harris.anderson" phot
4          <name>
5              <given>Harris</given>
6              <family>Anderson</family>
7          </name>
8          <email>harris.anderson@exampl
9          <link subordinates="robert.ta
10         <url href="http://www.example
11     </person>
12     <person id="robert.taylor" photo=
13         <name>
14             <given>Robert</given>
15             <family>Taylor</family>
16         </name>
17         <email>robert.taylor@example.
18         <link manager="harris.anderso
19         <url href="http://www.example
```

JSON XML(eXtensible Markup Language)

4. json Data Dictionary / List Conversion

4.3. Definition of json

- JSON (JavaScript Object Notation) was developed to deliver data object that consists of “key-value pair.”
- This data is in an open standard format that uses the text that we can read. It is a major data format for communications between the browser and server.
- Especially, it is widely used to express the data when exchanging data on the Internet.
- There are barely any restrictions in data type, and it is especially appropriate to express variables of a computer program.
- Originally, it was derived from JavaScript language, so it follows the JavaScript syntax, but it has language independent data format.
- Thus, since it has independent programming language or platform, the codes for syntax analysis and JSON data creation can be easily used in many different programming languages including C, C++, C#, Java, JavaScript, Perl, Python, etc.

4. json Data Dictionary / List Conversion

4.4. json data example

The following shows an example of JSON file. The following JSON includes personal information such as name, age, etc.

```
{
“Name": “David Doe",            → key: The string is used as the key and value in the form of value
“Age": 25,                                  → Use numbers (integers) as value
“Hobby": [“basketball“]          → List can be used as value
“Family": {“father": “John Doe", “mother": “Marry Doe"}, -> Can use the dictionary
“Married": true                          → Can use the Boolean data type
}
```

4. json Data Dictionary / List Conversion

4.5. Example code to convert json data into dictionary

- The json module provides various functions to read json type files or strings as dictionary and convert them into strings.
- To read the json type string, the loads function is used. The following is the process of reading and checking dictionary type personal data information in json data type. The json type data consists of strings can be converted into dictionary type through the loads function in the json module.

```python
import json

data = '{"Name": "David Doe", "Age": 25, "Hobby": ["basketball"],\
"Family": {"father": "John Doe", "mother": "Marry Doe"}, "Married": true}'

json_data = json.loads(data)

print(type(json_data))
print(json_data['Name'])
print(json_data['Family'])
print(json_data['Married'])
```

```
<class 'dict'>
David Doe
{'father': 'John Doe', 'mother': 'Marry Doe'}
True
```

4. json Data Dictionary / List Conversion

4.5. Example code to convert json data into dictionary

- The json module of Python provides various functions that read the json type files or strings as dictionary and convert them into strings. When using the json module in a code, use the “import json” command.
- The following code changes the novel information of The Old man and The Sea written by Ernest Hemingway into json type and reads it in the same way as the previous page.

```
1 import json
2
3 data = '{"title": "The Old man and The Sea", "ISBN": "12345", "Author" :"Ernest Hemingway"}'
4 json_data = json.loads(data)
5
6 print(type(json_data))
7 print(json_data['title'])
8 print(json_data['ISBN'])
9 print(json_data['Author'])
```

```
<class 'dict'>
The Old man and The Sea
12345
Ernest Hemingway
```

Line 4

- Convert the string type data into json type by using the loads function in the json module.
- The json_data object becomes dictionary data type.

4. json Data Dictionary / List Conversion

4.6. Example code to convert dictionary data type into json

Use the ”dump” function in the json module to write dictionary data type as json type file.

```
1 import json
2
3 data = '{"title": "The Old man and The Sea", "ISBN": "12345", "Author": "Ernest Hemingway"}'
4 json_data = json.loads(data)
5
6 # Code to create son data as book.json file
7 with open('book.json', 'w') as f:
8     json.dump(json_data, f, indent='\t')
```

Line 7,8

- Use the with() and open function to create the book.json text data in the writing mode (’w’).
- Use the json.dump command to record the json_data in the file “f” in json type.
- Indent by using indent = '\t’

4. json Data Dictionary / List Conversion

4.7. Recording the json data in a file

- The book.json file is created in the source code path of Jupyter Notebook.
- Open the file and there's json type file as shown below.
- The tab characters are used to indent.

File created by the json.dump function

Content of the book.json file: The text type json file has a similar syntax with Python dictionary.

4. json Data Dictionary / List Conversion

4.8. with ~ as syntax

- The with ~ as syntax will be discussed in detail in Unit 13. Let's learn about it briefly for now.
- When performing file processing in Python, file opening may fail. So, it is desirable to always conduct the exception test.
- Also, after finishing a given task after opening the file, make sure to close the file.
- The file processing can be conveniently done by with "with" syntax. The "with" syntax is another method for easy processing of exception while at the same time making clear syntax.

Read the book.json file with the open function in the writing mode (w).

After successfully reading the file, refer to this file as file type "f."

```
# Code to create json_data as book.json file
with open('book.json', 'w') as f:
    json.dump(json_data, f, indent='\t')
```

Record the json_data in the file "f."
For easier reading, indent using the tab character (\t).

Paper coding

Try to fully understand the basic concept before moving on to the next step.

Lack of understanding basic concepts will increase your burden in learning this course, which may make you fail the course.

It may be difficult now, but for successful completion of this course we suggest you to fully understand the concept and move on to the next step.

Paper coding

Q1. Create the capital_dic dictionary with the following string key-value items. Then, use the capital_dic to write results regarding Korea in the following dictionary items.

Program Execution Variable Declaration	key : Korea value : Seoul key : China value : Beijing key : USA value : Washington DC
Time	5 min

Output example

```
Seoul
```

Write the entire code and the expected output results in the note.

Paper coding

Q2. Create the fruits_dic dictionary that has elements of the following key-value items. Then, use this dictionary to print the price of each fruit as shown below.

Conditions for Execution	`The price of an apple is 5000 KRW.`
	`The price of a banana is 4000 KRW.`
	`The price of a grape is 5300 KRW.`
	`The price of a melon is 6500 KRW.`

Time	5 min

`fruits_dic dictionary`

key: apple value: 5000	key: banana value: 4000	key: grape value: 5300	key: melon value: 6500

Write the entire code and the expected output results in the note.

Unit 11. Dictionary Data Types

Let's code

1. Dictionary

1.1. Structure of dictionary

- Save the data as a dictionary with key and value.
  - Similar to the list, dictionary is a data structure to save vales, and it is a basic data type in Python.
  - However, a big difference with the list is that dictionary has a key related to the value.

1. Dictionary

1.2. Dictionary declaration

- The empty list is created with [ ], while the dictionary is created with { }.
- There are many ways to create a dictionary, but first use { } to create an empty dictionary and then add phone numbers one by one.

```
phone_book = { } # Create an empty dictionary
```

```
phone_book["David"] = "010-1234-5678"
```

1. Dictionary

1.3. How to operate dictionary

- Print the data that is created so far.

```
print(phone_book)
```

```
{'David': '010-1234-5678'}
```

- Dictionary can be created and initialized at the same time.

```
phone_book = {"David": "010-1234-5678"}
```

- Add few other phone numbers in the dictionary, and the following is the printed result.

```
phone_book["John"] = "010-1234-5679"
phone_book["Daniel"] = "010-1234-5680"
print(phone_book)
```

```
{'David': '010-1234-5678', 'John': '010-1234-5679', 'Daniel': '010-1234-5680'}
```

- When printing the phone_book dictionary, the items of the dictionary will be distinguished with commas.

2. Comparison of the list and dictionary

2.1. List and dictionary comparison chart

List	Dictionary
The index of the first value starts with 0, which is automatically assigned.	The designated key and value make a pair.
ex) lst = [11, 22, 33, 44, 55]	ex) dic = {0:11, 1:22, 2:33, 3:44, 4:55}

2. Comparison of the list and dictionary

2.1. List and dictionary comparison chart

Deletion with using "pop" in the list	Deletion with using "pop" in the dictionary
After calling lst.pop(0) lst = [22, 33, 44, 55]	After calling dic.pop(0) dic = {1:22, 2:33, 3:44, 4:55. 5:66}

2. Comparison of the list and dictionary

2.2. How to delete an item in the list

- See how the lst[1] changes after deleting a list item with the pop method.

```
lst = [11, 22, 33, 44, 55]
print('before pop(0) :', lst)
print('before pop(0) lst[1] = ', lst[1])
lst.pop(0)     # Use index 0 to delete the first item of the list
print('after pop(0) :', lst)
print('after pop(0) lst[1] = ', lst[1])   # The value referred by lst[1] changes
```

```
before pop(0) : [11, 22, 33, 44, 55]
before pop(0) list[1] = 22
after pop(0) : [22, 33, 44, 55]
after pop(0) list[1] = 22
```

- If an element is deleted in the list, then the referred value changes.

2. Comparison of the list and dictionary

2.2. How to delete an item in the list

- Compare the item deletion in the list and dictionary.
- See the dictionary item deletion and dic[1] change.
- The items method in the following code is a function that returns a tuple pair (key:value), which will be learned in detail later.

```
dic = {0:11, 1:22, 2:33, 3:44, 4:55}
# Print the items() function that returns the tuple pair (key-value) of dictionary
print('before pop(0) :', dic.items())
print('before pop(0) dic[1] = ', dic[1])
dic.pop (0)     # Items (0, 11) are deleted with the key 0
print('after pop(0) :', dic.items())
print('after pop(0) dic[1] = ', dic[1])    # The value referred by dic[1] does not change
```

```
before pop(0) : dict_items([(0, 11), (1, 22), (2, 33), (3, 44), (4, 55)])
before pop(0) dic[1] =  22
after pop(0) : dict_items([(1, 22), (2, 33), (3, 44), (4, 55)])
after pop(0) dic[1] =  22
```

- Even if an item is deleted, the value referred by the key does not change in dictionary.

2. Comparison of the list and dictionary

2.3. Create a list and dictionary and check item values

Classification	List (uses indices)	Dictionary (uses keys)
Check the item value	lst[0]	dic['one']

Classification	List	Dictionary
Create an empty object	lst = []	dic = {}
Create an object with items	lst = [1, 2, 3]	dic = {'one':1, 'two':2}

2. Comparison of the list and dictionary

2.4. Calculating the number of items in the list and dictionary and delete values

How to check the number of items in the list and dictionary and delete values

Classification	List	Dictionary
Check the numbers	len(lst)	len(dic)
Delete	del(lst[0]) or del lst[0]	del(dic['one']) or del dic['one']
Delete all	lst.clear()	dic.clear()

How to check if a specific value is included

Classification	List	Dictionary
Check the value	2 in lst	'one' in dic.keys 'one' in dic

3. Formatting for better print

3.1. format as print method

- Different prints have been done so far, but the distances between printed messages were not even.
- Learn about print methods for neat printing of the strings. The "format" method is extremely useful to print strings. Use the place holder as follows to print letters or numbers in an appropriate format.
- The {} below is the place holder, which designates the print location of the value that is the format method argument.

```
'{} Python!'.format('Hello')
```

```
'Hello Python!'
```

```
'I Like {} and {}'.format('Python', 'Java')
```

```
'I Like Python and Java'
```

3. Formatting for better print

3.2. Index for print

- Use indices including 0, 1, 2 regarding the format method argument to designate argument values.
- In the {0} place holder below, the first argument is printed due to the order of the format method arguments.
- In the {1} place holder below, the second argument is printed due to the order of the format method arguments.

```
'I like {0} and {1}'.format('Python', 'Java')
```

```
'I like Python and Java'
```

```
'I like {1} and {0}'.format('Python', 'Java')
```

```
'I like Java and Python'
```

```
'I like {0}, {0}, {0}'.format('Python', 'Java')
```

```
'I like Python, Python, Python'
```

- When using {0:4d}, four digits are used to print an integer, while {0:5d} results in five digits and {0:6d} six digits.

```
print('12345678901234567890')
print('{0:4d},{0:5d},{0:6d}'.format(123))
```

```
12345678901234567890
 123,  123,   123
```

3. Formatting for better print

3.3. Field width designation for neat printing

- It is also possible to print two decimal places with {0:.2f}. Designate {0:.3f} if printing to three decimal places.
- "f" refers to the floating point. If so, the width of the entire field can be designated in the front of the period as {0:5.2f} or {0:6.2f}.

```
1  'Precision of two decimal places: {0:.2f}. Precision of three decimal places {0:.3f}'.format(1/3)
```

```
'Precision of two decimal places: 0.33. Precision of three decimal places 0.333'
```

```
1  'Real numbers for 10 spaces : {0:10.3f}, {0:10.4f}'.format(3.1415926)
```

```
'Real numbers for 10 spaces :      3.142,     3.1416'
```

3. Formatting for better print

3.3. Field width designation for neat printing

- For printing integers, it is possible to designate the number of spaces for printing such as {1:3d}, {1:4d}, {1:5d}.
- Create a code to print three integers by designating 3, 4, and 5 spaces respectively to three different arguments.
- The result would be the nice and neat print as shown below.

```
for i in range(2, 11, 2):
    print('{0:3d} {1:4d} {2:5d}'.format(i, i*i, i*i*i))

```

```
  2    4     8
  4   16    64
  6   36   216
  8   64   512
 10  100  1000
```

Pair programming

Pair Programming Practice

Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.

Pair Programming Practice

Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

Collaborative learning environment

A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor's question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students "divide and conquer." When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

Pair programming

Q1. Create the fruits_dic dictionary consists of key-value pairs including ('apple', 6000), ('melon', 3000), ('banana', 5000), ('orange', 4000). Then, print all of the key in the fruits_dic as list type and examine if the 'apple' and 'mango' keys are found in the fruits_dic, and print as follows.

Output example

```
dict_keys(['apple', 'melon', 'banana', 'orange'])
apple is in fruits_dic.
mango is not in fruits_dic.
```

Unit 12.

Sequence Data Types

Unit learning objective (1/3)

Learning objectives

- ✓ Be able to find and apply a specific value in the sequence data type.
- ✓ Be able to integrate two objects through linking sequence objects.
- ✓ Learn iteration data types from sequence objects and be able to print using iteration statements.
- ✓ Be able to calculate the number of certain elements of the sequence object.
- ✓ Be able to draw a specific range by using the sequence object index.
- ✓ Learn the len function that is frequently used in sequence objects and be able to use the range data type.

Unit learning objective (2/3)

Learning overview

- ✓ Learn the searching functions of each string, tuple, and list data type objects.
- ✓ Learn how to link sequence objects and calculate the number of specific elements within the sequence object.
- ✓ Understand the types and characteristics of iteration data types and learn how to appropriately use them.
- ✓ Learn how to use indices of sequence objects.
- ✓ Learn how to calculate the length of sequence objects using the len function.

Concepts you will need to know from previous units

- ✓ Be able to accurately understand the concepts of the list, range, tuple, and string which are sequence objects and declare them.
- ✓ Be able to use frequently used operators of sequence objects including in, not in operators, etc.
- ✓ Be able to add and delete elements to the list and perform slicing and indexing.

Unit learning objective (3/3)

Keywords

- Sequence object
- Iteration
- Searching, linking
- Finding a specific element
- Index
- len, range

Mission

1. Real world problem

1.1. Aging issue

http://www.inhapress.com/news/articlePrint.html?idxno=6982

- Aging is a serious social problem in countries like Korea and Japan.
- Aging refers to the percentage of elderly population in the total population.
- Major problems of aging are as follows.
  - Poverty due to loss of workforce and lack of making provision for old age
  - Increased financial burden of young and middle-aged people due to providing care for the elderly
  - Competition between young people and the elderly regarding work
  - Neglected old people due to generalization of nuclear families
- Investigate and calculate the aging percentage in your town.

1. Real world problem

1.2. Aging percentage calculator

http://www.inhapress.com/news/articlePrint.html?idxno=6982

- There is a tuple that shows population composition of a town. In this tuple, the first element is the number of population between 0 to 9 years old, and the second element is the population between 10~19 years old, which is teenage population. The third element is the population of 20s including 20~29 years old people, and the i+1 element has number of population between 10i~10i+9 years old. 11th element is the last one, which includes populations over 100 years old.
- Assume that the tuple for population composition of towns A and B is as follows.

```
population_a = (100, 150, 230, 120, 180, 100, 140, 95, 81, 21, 4)
population_b = (300, 420, 530, 420, 400, 300, 40, 5, 1, 1, 1)
```

- Compare the degree of aging of each town. Suppose that the degree of aging is the percentage of the elderly who are 70 years old and older in the total population. Write a program to make the following results. (Use slicing to extract necessary parts from the data.)

1. Real world problem

1.3. Additional data related with aging

- For more details about aging, refer to https://m.koreatimes.co.kr/pages/article.asp?newsIdx=270287.

https://m.koreatimes.co.kr/pages/article.asp?newsIdx=270287

2. Mission

2.1. How the aging percentage calculator works

```
population_a = (100, 150, 230, 120, 180, 100, 140, 95, 81, 21, 4)
population_b = (300, 420, 530, 420, 400, 300, 40, 5, 1, 1, 1)
oldA = sum(population_a[7:])
oldB = sum(population_b[7:])
sumA, sumB = sum(population_a), sum(population_b)

oldRateA, oldRateB = oldA/sumA, oldB/sumB

print('The degrees of aging in town A and B are {:5.3f} and {:5.3f} respectively.'.format(oldRateA, oldRateB))
```

```
The degrees of aging in town A and B are 0.165 and 0.003 respectively.
```

2. Mission

2.2. Programming plan

Pseudocode

[1] Start

[2] Enter the population data of the town.

[3] Use slicing to only add the elderly who are 70 years old or older

[4] Add the total population of the town

[5] Divide the elderly population with the total population of the town.

[6] Print the aging percentage.

[7] End

Flowchart

2. Mission

2.3. Aging percentage calculator final code

```
population_a = (100, 150, 230, 120, 180, 100, 140, 95, 81, 21, 4)
population_b = (300, 420, 530, 420, 400, 300, 40, 5, 1, 1, 1)
oldA = sum(population_a[7:])
oldB = sum(population_b[7:])
sumA, sumB = sum(population_a), sum(population_b)

oldRateA, oldRateB = oldA/sumA, oldB/sumB

print('The degrees of aging in town A and B are {:5.3f} and {:5.3f} respectively.'.format(oldRateA, oldRateB))
```

Key concept

1. Searching a Specific Value

1.1. Sequence Data Type

- The list, tuple, range, and string have one thing common, which is that multiple values are saved in a sequence inside those data structures.
- In Python, the data types with sequence of values such as list, tuple, range, string are called sequence types.
- The objects created from sequence data types are called sequence objects, and each value in the sequence objects is called an element.

Example of sequence data types

1. Searching a Specific Value

1.2. Member operators: in, not in

- The member operators “in” and “not in” are operators that return True or False.
- Member operators are used to check if a specific element is included in data structures such as string, list, tuple.
- in operator: Returns True if the member is present, and False if not.
- not in operator: Returns False if the member is present, and True if not.
- From the example below, because list1 has 10, 10 in list1 returns True, while 10 not in list1 returns False.

```
list1 = [10, 20, 30, 40]
10 in list1
```

```
True
```

```
list1 = [10, 20, 30, 40]
10 not in list1
```

```
False
```

1. Searching a Specific Value

1.3. Using the "in" operator

- When searching for a specific value in tuple, use the "in" operator.

```
tup = (1, 2, 3, 4)
3 in tup
```

```
True
```

- Use the "in" operator in the range type sequence created by the range function.

```
11 in range(10)
```

```
False
```

- Use the "in" operator to search a specific value of string. The following is True because 'a' is included in 'abcd.'

```
'a' in 'abcd'
```

```
True
```

2. Linking Sequence Objects

2.1. Linking sequence objects

- For sequence data types, it is possible to use the + operator.
- However, range cannot be linked with the + operator.
- Linking lists

```
list1 = [11, 22, 33, 44]
list2 = [55, 66]
print(list1)
print(list1 + list2)
```

```
[11, 22, 33, 44]
[11, 22, 33, 44, 55, 66]
```

2. Linking Sequence Objects

2.2. Linking sequence objects

- This shows how to link tuples. Use the + operator.

```
tup1 = (1, 2, 3)
tup2 = (4, 5, 6)
print(tup1 + tup2)
```

```
(1, 2, 3, 4, 5, 6)
```

- This shows how to link strings. Use the + operator.

```
str1 = 'hello '
str2 = 'world'
print(str1 + str2)
```

```
hello world
```

2. Linking Sequence Objects

2.3. Possible error from linking sequence objects

TypeError

```
range (10) + range (10, 20)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[13], line 1
----> 1 range (10) + range (10, 20)

TypeError: unsupported operand type(s) for +: 'range' and 'range'
```

- Among the sequence data types, + operator cannot link the range objects.

If so, linking is possible by making the range into the list or tuple.

```
list(range(10)) + list(range(10,20))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
tuple(range(10)) + tuple(range(10,20))
```

```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19)
```

3. Iterating Sequence Objects

3.1. Iterating sequence objects

- Operators for iterating sequence object: *
  - Sequence data types can use a multiplication operator, which creates an object element repeatedly.
  - However, multiplication of objects is impossible for range with the * operator.
  - Written as [sequence data type] * integer

- List iteration

```
list1 = [11, 22, 33, 44] * 2
print(list1)
```

```
[11, 22, 33, 44, 11, 22, 33, 44]
```

3. Iterating Sequence Objects

3.1. Iterating sequence objects

Tuple iteration

```
tup1 = (1, 2, 3)
print(tup1 * 2)
```

```
(1, 2, 3, 1, 2, 3)
```

String iteration

```
str2 = 'hello'
print(str2 * 3)
```

```
hellohellohello
```

3. Iterating Sequence Objects

3.2. Cautions for iterating sequence objects

Error that occurs from using the * operator to the range type: TypeError

```
range(10) * 3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[21], line 1
----> 1 range(10) * 3

TypeError: unsupported operand type(s) for *: 'range' and 'int'
```

- For range data type, objects cannot be linked by using the + operator.
- Similarly, the range data type cannot be iterated by using the * iteration operator.

3. Iterating Sequence Objects

3.3. Conversion of the type for range data type iteration

- The range data type cannot be iterated with the * operator.
- Thus, it should be converted into the list or tuple data type as shown below.

```
ran = list(range(5)) * 3
print(ran)
```

```
[0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4]
```

```
ran = tuple(range(5)) * 3
print(ran)
```

```
(0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4)
```

3. Iterating Sequence Objects

3.4. Caution for using the * operator.

*** operation of two lists that induce syntactic error**

```python
list1 = [11, 22, 33, 44]
list2 = [55, 66]
print (list1 * list2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[24], line 3
      1 list1 = [11, 22, 33, 44]
      2 list2 = [55, 66]
----> 3 print (list1 * list2)

TypeError: can't multiply sequence by non-int of type 'list'
```

- The [sequence data type] * [sequence data type] operation cannot be done between the list, tuple, and string data types.
- All of the sequence data types should use [sequence data type] * [integer] for iterative operation.

4. Calculating the Number of Specific Sequence Object Elements

4.1. count method

Learn about the count method to calculate the number of specific elements in the sequence.

- The count method returns the number of elements in the sequence object.
- In the code below, three 11 are found in the list1, so it returns 3.

```
list1 = [11, 11, 11, 22, 33, 44]
print(list1.count (11))        # Number of specific elements
```

```
3
```

4. Calculating the Number of Specific Sequence Object Elements

4.1. count method

- Printing the count method in tuple
  - There are three 11 in the tup1.

```
tup1 = (11, 11, 11, 22, 33, 44)
print(tup1.count (11))        # Number of specific elements
```

```
3
```

- Printing the count method in string
  - There are three 'l' in the str1.

```
str1 = 'hello world'
print(str1.count('l'))        # Number of specific elements
```

```
3
```

4. Calculating the Number of Specific Sequence Object Elements

4.2. count method example code

Compare the len and count print results of the range data type 'ran'.

```
1 ran = range(0, 5, 1)
2 print(len(ran))
3 print(ran.count(2))
```

```
5
1
```

Line 2, 3

- ran has five elements of 0, 1, 2, 3, 4. The number of elements printed by using the len function would be 5.
- On the other hand, when using the count method to see how many 2s are in the ran variable, it will print 1 because there is only one 2.

5. Using Sequence Object Index

5.1. Index and sequence objects

- Index is used in sequences objects in common.
- Index:
  - Index refers to a number that indicates an object value of a list or sequence.
  - The indices of a list with n items increase from 0 to n-1.
- Index of a sequence object always starts from 0.

In sequence data types, indices are used to refer to an element.

5. Using Sequence Object Index

5.2. Indexing sequence objects

- Element value reference through list indexing

```
list1 = [11, 11, 11, 22, 33, 44]
```

```
list1[0]
```

```
11
```

```
list1[2]
```

```
11
```

- Element value reference through string indexing

```
str1 = 'hello world'
```

```
str1[2]
```

```
'l'
```

```
str1[7]
```

```
'o'
```

5. Using Sequence Object Index

5.2. Indexing sequence objects

- Element value reference through tuple indexing

```
tup1 = (1,1,1,2,3,3,4)
```

```
tup1[3]
```

```
2
```

- Element value reference through range indexing: In the range type, a step value such as 1 in (0, 5, 1) can be used to designate the jump space, which will be explained later.

```
ran = range(0,5,1)
```

```
ran[4]
```

```
4
```

5. Using Sequence Object Index

5.3. Slicing sequence objects

- In general, the sequence data type slicing specifies the start and end indices.
- All of the sequence data types use the same slicing method as provided in the rule below.
- [1:5] range designation brings elements from index 1 to index 5-1.
- Thus, the elements [20, 30, 40, 50] are sliced.

5. Using Sequence Object Index

5.3. Slicing sequence objects

- When importing until the end of the entire sequence, last index can be omitted.
- [1:] slicing can slice from the 2nd to last element.

```
[1:] = [20, 30, 40, 50, 60, 70, 80]
```

5. Using Sequence Object Index

5.3. Slicing sequence objects

- The start and end indices can be omitted as well.
- When used together with [:], it imports all items.

5. Using Sequence Object Index

5.4. Negative indexing of sequence objects

- Negative indices can be used for sequence objects.
- The index of the last element becomes -1, and -2, -3, ... are given to the front elements.

5. Using Sequence Object Index

5.4. Negative indexing of sequence objects

- The negative index rule is applied to string objects.

string object element	'a'	'b'	'c'	'd'	'e'	'f'
Negative index	[-6]	[-5]	[-4]	[-3]	[-2]	[-1]

```
str1 = 'abcdef'
print(str1[-1])
```

```
f
```

```
print(str1[-6])
```

```
a
```

5. Using Sequence Object Index

5.4. Negative indexing of sequence objects

- The negative index rule is applied to tuple objects.

tuple object element	11	22	33	44	55	66	77
Negative index	[-7]	[-6]	[-5]	[-4]	[-3]	[-2]	[-1]

```
tup1 = (11, 22, 33, 44, 55, 66, 77)
print(tup1[-1])
```

```
77
```

```
print(tup1[-6])
```

```
22
```

5. Using Sequence Object Index

5.5. Negative indexing of sequence objects

- The negative index rule is applied to range objects.

range object element	1	2	3	4	5	6
Negative index	[-6]	[-5]	[-4]	[-3]	[-2]	[-1]

```
ran = range(1,7)
print(ran[-6])
```

```
1
```

```
print(ran[-1])
```

```
6
```

5. Using Sequence Object Index

5.6. Slicing using negative indices

- The following is the slicing method using negative indices.
- [-7:-2]: Includes items [20, 30, 40, 50, 60] when using the negative index range.

5. Using Sequence Object Index

5.6. Slicing using negative indices

- In case of using negative indices, it will import until the last item of the list when omitting the last index as shown below.

5. Using Sequence Object Index

5.6. Slicing using negative indices

- It is possible to skip the first index in slicing that uses negative indices.
- [:-2] = imports the value until the -3 index

6. Applications of lens and range

6.1. range example code

Create the even_list that has even numbers between 1 to 10 as its elements (including 10). Use the print function to print the list as follows.

```
even_list = list(range(2, 11, 2))
print('even_list =', even_list)
```

```
even_list = [2, 4, 6, 8, 10]
```

- The sequence created by the range function is converted to the list through the list function.
- Because the last argument of range is 2, increase by adding 2 from 2 to 10. As a result, the elements are 2, 4, 6, 8, 10.
- Lists with various numbers can be easily created by converting the range into list type.

```
list(range(10))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
list(range(2, 10))
```

```
[2, 3, 4, 5, 6, 7, 8, 9]
```

```
list(range(2, 10, 3))
```

```
[2, 5, 8]
```

6. Applications of lens and range

6.2. len example code

- Create a list named ”nations” that has ‘Korea', 'China', 'Russia', 'Malaysia’ as its elements. Then, print the last element of the list by using the len(nations)-1 index.

```
nations = ['Korea', 'China', 'Russia', 'Malaysia']
print('last element of nations :', nations[len(nations)-1])
```

```
last element of nations : Malaysia
```

- Use country names as the elements of the list.
- Approach to the last index by subtracting 1 from the length obtained by len function.

6. Applications of lens and range

6.3. ASCII code of the letter

- If there is tuple a that has elements of ('A','B','C') and tuple b with elements of ('A','B','D'), then the code value of C is 67 and D is 68. So, a > b is False and a < b is True. It is determined by comparing the size of the tuple elements.

```
a = ('A','B','C')
b = ('A','B','D')
```

```
ord('C')
```

```
67
```

```
ord('D')
```

```
68
```

```
a > b
```

```
False
```

```
a < b
```

```
True
```

One More Step

What is the ASCII code used in strings?

- ASCII code is an abbreviation of 'American Standard Code for Information Interchange.' It designates and manages a number for each character and is the most fundamental character code.
- In early ages of computer development, the method to express a letter 'A' differed in each computer company. For instance, company A used 120 and company B used 45 to express a letter 'A.'
- As the computer industry made developments, it was necessary to convert characters and symbols in the same number expression to communicate with other programs or computers. So, ASCII code was developed, which is a seven-bit code that assigns 128 numbers into alphabets, numbers, special letters, control characters, etc.

Paper coding

Try to fully understand the basic concept before moving on to the next step.

Lack of understanding basic concepts will increase your burden in learning this course, which may make you fail the course.

It may be difficult now, but for successful completion of this course we suggest you to fully understand the concept and move on to the next step.

Q1. Predict the execution result of the following code and provide handwritten result.

Conditions for Execution	Predict the result of the following program and provide handwritten coding results.
Time	5 min

Output example

```
t1 = 'a', 'b', 'c'
t2 = ('a', 'b', 'c')
t3 = ('d', 'e')

print(t1 == t2)

print(t1 > t3)

print(t1 < t3)

print(t2 + t3)

print ([ t2 + t3 ])

print(t1)
```

Write the entire code and the expected output results in the note.

Paper coding

Q2. The following tuple records daily sales of a store for 10 days. Write a code to print how many days had reduced sales compared to previous day. (Hint: compare the values by iterating the elements with the iteration statement.)

Conditions for Execution	`Daily sales record: (100, 121, 120, 130, 140, 120, 122, 123, 190, 125)` `In the past 10 days, 3 days had reduced sales compared to the previous day.`
Time	10 min

Write the entire code and the expected output results in the note.

Unit 12. Sequence Data Types

Let's code

1. Tuple is an immutable object among sequence objects

1.1. Definition of tuple

- Tuple is a collection data type that has multiple elements (item values).
- Collection data type refers to a data type that can have many items as elements such as the list, tuple, dictionary, and set.
- However, unlike the list, it is impossible to change the sequence of elements once designated.
- t = ('one', 'two', 'three')
- Immutable
  - Impossible to change or delete an object in the tuple

1. Tuple is an immutable object among sequence objects

1.2. How to create tuple

There are many different methods to create a tuple.

Creating an empty tuple	tuple0 = tuple() #empty parenthesis must be used
Creating a tuple with one element	tuple1 = (1,) #must use a comma
Creating a basic tuple using parenthesis	tuple2 = (1, 2, 3, 4)
Creating a simple tuple	tuple3 = 1, 2, 3, 4
Creating a tuple from a list	n_list = [1, 2, 3, 4] tuple4 = tuple(n_list)

1. Tuple is an immutable object among sequence objects

1.3. Integer type tup variable

- Cautions for tuple declaration with one element: integer type tup variable
- When trying to create a tuple with one item, if you allocate tup = (100), it is considered the same as tup = 100, so the tup becomes an integer, not a tuple. So, make sure to insert a comma like tup = (100,) when using tup as tuple.

```
1 tup = (100)
2 print(tup)
3 print(type(tup))
```

```
100
<class 'int'>
```

Line 1

- Entering tup = (100) will result in tup = 100.
- The data type of tup becomes int type.

1. Tuple is an immutable object among sequence objects

1.4. Tuple type tup variable

When trying to create a tuple with one item, if you allocate tup = (100), it is considered the same as tup = 100, so the tup becomes an integer, not a tuple. So, make sure to insert a comma like tup = (100,) when using tup as tuple.

```
1 tup = (100)
2 print(tup)
3 print(type(tup))
```

```
100
<class 'int'>
```

Line 1

- Entering tup = (100,) results tup a tuple type.
- The data type of tup becomes tuple type.

1. Tuple is an immutable object among sequence objects

1.5. Differences between tuple and list

- Unlike the list, internal values of a tuple cannot be changed. So, assignment operation for individual item as shown below will result in an error.
- Such property is called immutable.

TypeError

```
t = (0, 1, 2, 3, 4)
t[0] = 100
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-78-9d561bf19fbf> in <module>
      1 t = (0, 1, 2, 3, 4)
----> 2 t[0] = 100

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

1. Tuple is an immutable object among sequence objects

1.6. Packing and unpacking

- Packing: Refers to adding many values to one variable.
- Unpacking: Refers to taking many values from a packed variable.

Packing	Unpacking

`a = (1, 2)  # Tuple packing`
`a[0]  # Reference for tuple item`
 	
1
	`c = (3, 4)  # Tuple packing`
	`x, y = c  # Assigning to 2 variables by unpacking tuple c`
	`x`
	
	3

`a[1]` 2	`y` 4

1. Tuple is an immutable object among sequence objects

1.7. Swap

- a is allocated with 100 and b is allocated with 200, and the mission is to swap the values of two variables.
- The code below is a swap method that is generally used in C or Java language.
- For swapping, a temporary variable temp is used for three assignment statements.
- Python uses a much simpler method.

```
a = 100
b = 200
print('before swap : a = ', a, 'b = ', b)
temp = a
a = b
b = temp
print('after swap : a = ', a, 'b = ', b)
```

```
before swap : a = 100 b = 200
after swap : a = 200 b= 100
```

1. Tuple is an immutable object among sequence objects

1.7. Swap

- Use Python tuple to easily create the code in the previous page.
- In Python, swapping is possible by using a single line a, b = b, a as shown below.
- The code is very simple and intuitive.

```
a = 100
b = 200
print('before swap : a = ', a, 'b = ', b)
a, b = b, a   # very simple swap
print('swap result using tuple: a = ', a, 'b = ', b)
```

```
before swap : a = 100 b = 200
swap result using tuple : a = 200 b = 100
```

1. Tuple is an immutable object among sequence objects

1.8. How to sort tuple

- Because elements cannot be changed in tuple, using the sort method is not allowed.
- When sorting tuple elements, convert the tuple into a list and use sort method to sort the numbers.

```
tup = (1, 2, 5, 4, 3, 2, 9, 3, 7, 3, 9)
temp = list(tup)
temp.sort()
print(temp)
```

```
[1, 2, 2, 3, 3, 3, 4, 5, 7, 9, 9]
```

Pair programming

Pair Programming Practice

Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.

Pair Programming Practice

Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

Collaborative learning environment

A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor's question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students "divide and conquer." When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

Pair programming

Q1. Return the element with the maximum number of occurrences. When there are more than two frequent elements, print the highest number.

Output example

```
Given tuples: (1, 2, 5, 4, 3, 2, 1, 4, 7, 8, 9, 9, 3, 7, 3, 9)
The most frequent element: 9
```

Pair programming

Q2. In the output example below, there are the tuples containing elements, as well as empty tuples, empty strings and empty lists that have no elements.
Write a code to remove these empty tuples, empty strings and empty lists from the given list below.
(However, do not remove (,) tuple because it is considered as having one empty tuple.)

Output example

```
Given tuples: [(), (1,), [], 'abc', (), (), (1,), ('a',), ('a', 'b'), ((),), '']
The most frequent element: [(1,), 'abc', (1,), ('a',), ('a', 'b'), ((),)]
```

Unit 13.

Two-Dimensional Lists

Unit learning objective (1/3)

Learning objectives

- ✓ Be able to create two-dimensional lists and access elements in the lists using index.
- ✓ Be able to assign values in the list elements and print those.
- ✓ Understand two-dimensional array while creating, assigning, duplicating two-dimensional list using loop statement.
- ✓ Understand the features of Python's two-dimensional list and be able to create jagged lists.

Unit learning objective (2/3)

Learning overview

- ✓ Create two-dimensional lists and access each element using index.
- ✓ Print values in the two-dimensional list using loop statement.
- ✓ Create jagged lists using loop statement.
- ✓ Use contraction expressions to efficiently address two –dimensional lists.

Concepts you will need to know from previous units

- ✓ Creating and connecting sequence objects
- ✓ Accessing a particular element using sequence object indexing
- ✓ Using double loop statement and key methods of sequence objects

Unit learning objective (3/3)

Keywords

- Two-Dimensional Lists
- Two-Dimensional Indexing
- Jagged Lists
- Double Loop Statement

Mission

1. Real world problem

1.1. Downturn in theater industry due to the development of OTT

- Today, due to the development of over-the-top media service (OTT) technology such as Netflix and YouTube, the theater industry is facing a big crisis.
- The theater industry makes a significant contribution to the local economy, so if local theaters are closed, the unemployment rate would increase.
- Currently, the theater industry is overcoming the crisis by continuously developing differentiated customer services and additional services.
- As part of the services, some operate small theaters with premium services like high-end recliner.

1. Real world problem

1.2. The movie theater seat booking system

- In this mission, we will create a theater seat booking system.
- Recently, the movie theater has a well-established booking system, so you can check which seats are available on your smartphone or on-site counter.
- Now you will create a seat notification system available in a small theater with 18 seats in 3 rows and 6 columns as shown in the figure.
- The system should have the function of creating empty seats and reserved seats randomly and telling the number of empty seats.

1. Real world problem

1.3. Example of movie theater seat booking system

The number of empty seats: 5

- For simulation, a two-dimensional list should be created and marked as 0 if the seat is empty and 1 if not.
- The seats in this movie theater consist of a two-dimensional list. It is a small movie theater and assumed that it consists of a total of 18 seats in 3 rows and 6 columns.
- The empty and reserved seats are randomly determined through the random module. The seats are given a value of 1 or 0.
- Let's create a program that prints the location and number of empty seats.

1. Real world problem

1.4. Additional reference for movie theater seat booking

- If you want to know about a movie theater seat booking example, visit https://www.amctheatres.com/.
- AMC Theater is the world's largest theater chain and has tens of thousands of theater chains in over 15 countries.

2. Mission

2.1. How a movie theater seat booking system works

```python
import random

seat = []

for i in range(3):    # Randomly input occupied/unoccupied seats
    line = []
    for j in range (6):
        rand = random. randrange (0, 2) # Use random numbers of 0 and 1
        line.append(rand)
    seat.append(line)

available = 0

for i in range(3):    # Print the seats, and calculate the number of remaining seats
    for j in range(6):
        print(seat[i][j], '', end='')
        if seat[i][i] == 0:
            available += 1
    print()
print('The number of remaining seats in the theater:', available)
```

```
1 0 1 0 0 0
0 0 0 0 1 1
1 0 1 1 1 1
The number of remaining seats in the theater: 6
```

2. Mission

2.1. How a movie theater seat booking system works

- Since this code is somewhat long, it is divided into two flowcharts as below.
  - Flowchart 1: Algorithm that randomly generates empty seats in a movie theater
  - Flowchart 2: Algorithm to find the number of empty seats out of all seats

2. Mission

2.2. Programming Plan

Pseudocode 1: Generating random seats

```
[1] Start
[2] Reset the seat list.
[3] for i = 0 for three times do {
    [4] Reset the line list.
    [5] for j = 0 for six times do{
        [6] Generate random numbers between 0 and 1.
        Add those random numbers to the line list.
    [7] Add line to the seat list. }
}
[8] End
```

Flowchart 1: Generating random seats

2. Mission

2.2. Programming Plan

Pseudocode 2: Printing seat information

```
[1] Start
[2] Reset the number of empty seats to 0.
[3] for i = 0 to for three times do {
    [4] for j = 0 to for six times do{
        [5] Print seat[i, j].
        [6] if seat[i, j] == 0 {
            [7] Increase the number of empty seats by 1.
        }
}
[8] Print the number of empty seats.
[9] End
```

Flowchart 2 : Printing seat information

2. Mission

2.3. Final code of the move theater seat booking system

```python
import random

seat = []

for i in range(3):    # Randomly input occupied/unoccupied seats
    line = []
    for j in range (6):
        rand = random. randrange (0, 2) # Use random numbers of 0 and 1
        line.append(rand)
    seat.append(line)

available = 0

for i in range(3):    # Print the seats, and calculate the number of remaining seats
    for j in range(6):
        print(seat[i][j], '', end='')
        if seat[i][i] == 0:
            available += 1
    print()
print('The number of remaining seats in the theater:', available)
```

Key concept

1. Creating a Two-Dimensional List

1.1. Example of a two-dimensional list with 3 rows and 3 columns

- A two-dimensional list is a data structure in which one-dimensional list are grouped, and is widely used.
- In order to create a two-dimensional list, a list must be created in the list.
- The figure below is a two-dimensional list with 3 rows and 3 columns.

1. Creating a Two-Dimensional List

1.1. Example of a two-dimensional list with 3 rows and 3 columns

Create a two-dimensional list containing lists as elements in the list.

```
list_array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

- This is a 2D list of 3 x 3 size which is a set of three lists with three numbers.

```
list_array = [[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]]
```

- To make it easier to recognize a two-dimensional list, you can enter three horizontal and three vertical lines.
- These two methods are the same, but the second one might be easier to understand.

1. Creating a Two-Dimensional List

1.1. Example of a two-dimensional list with 3 rows and 3 columns

- Create a two-dimensional list containing elements inside the list as shown in Figure. list_array[0] refers to the first element, [1, 2, 3].

```python
list_array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
list_array[0]
```

```
[1, 2, 3]
```

2. Accessing Two-Dimensional List Elements

2.1. Indexing to access list elements

Access elements of a two-dimensional list using indices.

```
list_array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
list_array[0]
```

```
[1, 2, 3]
```

```
list_array[0][2]
```

```
3
```

- To access the elements of a two-dimensional list or allocate values to the elements, use [ ] (square brackets) twice after the list and specify a row index and a column index within [ ].

2. Accessing Two-Dimensional List Elements

2.2. Using indices to access 2D list, and read and allocate values

```
list_array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

- The indices of a 2D list of 3x3 size start at 0. Thus, the first horizontal and first vertical element of the list becomes list_array[0][0]. The last element becomes [2][2].

```
list_array[0][0]
```

```
1
```

```
list_array[2][2]
```

```
9
```

- Allocate 300 to the vertical index (row) 1 and the horizontal index (column) 1.

```
list_array[1][1] = 300
```

```
print(list_array[1][1])
```

```
300
```

3. Printing Two-Dimensional List Elements

3.1. Printing a two-dimensional list

- Print two-dimensional list which is somewhat tricky.
  - If you use the for statement with two-dimensional list, it will bring elements sequentially from the first.
  - For this reason, it is difficult to access individual elements such as 1, 2, and 3 in the following way.

```
list_array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for item in list_array:
    print('item =', item)
```

```
item = [1, 2, 3]
item = [4, 5, 6]
item = [7, 8, 9]
```

3. Printing Two-Dimensional List Elements

3.2. Printing a two-dimensional list using the for loop

- Using for with a two-dimensional list, return the lists of elements from the first to the last.
- If you specify three variables before 'in', such as for i, j, k in list_array:, the elements 1, 2, and 3 of the first list are called.
- That is, since the list [1, 2, 3] appears when running for loop, 1, 2, and 3 are assigned to i, j, and j, respectively.
- Repeat the above process to bring the entire items.

```python
list_array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for i,j,k in list_array:
    print(i,j,k)
```

```
1 2 3
4 5 6
7 8 9
```

In the code below, the first elements in list_array are assigned to i, j, k

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
for i,j,k in list_array:
    print(i,j,k)
```

3. Printing Two-Dimensional List Elements

3.3. Two-dimensional list and ValueError

- The method on the previous page only works properly when there are exactly three items in the list..
- Accordingly, an error occurs when there are not three list items in the list as below.

ValueError

```
list_array = [[1, 2, 3], [4, 5], [7]]
for i,j,k in list_array:
    print (i, j, k)
```

```
1 2 3

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[24], line 2
      1 list_array = [[1, 2, 3], [4, 5], [7]]
----> 2 for i,j,k in list_array:
      3     print (i, j, k)

ValueError: not enough values to unpack (expected 3, got 2)
```

3. Printing Two-Dimensional List Elements

3.4. Double for loop and two-dimensional list

- Try printing using double for loop as following.
- The outer for loop for i in list_array: assigns column by column from the entire list to i.
- Using the inner for j in i :, print each elements from i.

```
list_array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for i in list_array:
    for j in i:
        print(j, end = ' ')
    print()
```

```
1 2 3
4 5 6
7 8 9
```

3. Printing Two-Dimensional List Elements

3.5. Example of two-dimensional double for loop

- Try using double for loop as following.
- This method is a code that operates well even if the number of items in the list is different.

```
1 list_array = [[1, 2, 3], [4, 5], [7]]
2 for i in list_array:
3     for j in i:
4         print(j, end = ' ')
5     print()
```

```
1 2 3
4 5
7
```

Line 2, 3, 4

- Put one element in the list_array into i. The first element is [1, 2, 3].
- Put the element in i back into j.
- In this case, the initial j becomes 1, and 2 and 3 are assigned while going around the for loop, respectively.
- When end = ' ' is used, spacing is used instead of line breaking.

3. Printing Two-Dimensional List Elements

3.6. Printing using index

Focus Printing using index : calculating and printing the size of the 2D list with len.

- Put the size of the row and column in for in range and index them to print the elements in the list.
- Note here that len(list_array) is 3, not 9.
- And you have to find the size of the inner list, list_array[i], with len to obtain the size (number of columns) of the elements in the inner list.
- Thus, len(list_array[i]) is 3 in this code.

```
1 list_array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
2 for i in range(len(list_array)):
3     for j in range(len(list_array[i])):
4         print(list_array[i][j], end = ' ')
5     print()
```

```
1 2 3
4 5 6
7 8 9
```

Line 2, 3

- len(list_array) : Returns row size, 3.
- len(list_array[i]) : Returns the size of the element in row i.

3. Printing Two-Dimensional List Elements

3.7. Double for loop in case of two-dimensional list with different number of items

This method is also a code that operates well even if the number of items in the list is different.

```
1 list_array = [[1, 2, 3], [4, 5], [7]]
2 for i in range(len(list_array)):
3     for j in range(len(list_array[i])):
4         print(list_array[i][j], end = ' ')
5     print()
```

```
1 2 3
4 5
7
```

Line 2, 3, 4

- Repeat for loop three times using range(len(list_array)).
- j repeats a code in the loop for the the number of items in list_array[i].
- Print elements in list_array using [i][j] index.

4. Loop Statement and Two-Dimensional List

4.1. Two-dimensional list assignment and reference

- If list_array is assigned to list1 and then the value of list1 is changed, the value of list_array will be also changed.

```python
list_array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
list1 = list_array
print(list1)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```python
list1[0][0] = 90
print(list_array)
print(list1)
```

```
[[90, 2, 3], [4, 5, 6], [7, 8, 9]]
[[90, 2, 3], [4, 5, 6], [7, 8, 9]]
```

- Assignment operation in a list is not in a way of copying and passing values.
- Therefore, list_array and list1 refer to the same object.(This is called a reference)
- Since they refers to the same object, when the value of the object is changed, both lists export the changed value.

One More Step

To create the two in completely different memories, the copy.deepcopy function of the copy module must be used.

```
import copy
list_array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
list1 = copy.deepcopy(list_array)
list1[0][0] = 90
print(list_array)
print(list_array is list1)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
False
```

Result of copy.deepcopy : Objects are created in separate memory.

4. Loop Statement and Two-Dimensional List

4.2. Use for loop to create two-dimensional list of 3 rows and 2 columns

```
1 list1 = []
2 for i in range(3):
3     line = []
4     for j in range(2):
5         line.append(0)
6     list1.append(line)
7
8 print(list1)
```

```
[[0, 0], [0, 0], [0, 0]]
```

Line 2, 3

- First, create an empty list line to be used as an inner list by repeating for the size of the row, 3.

Line 4, 5, 6

- Then add 0 as an append to the line, repeating for the size of the column, 2.
- In outer loop, append is used again. In this way, an inner list line is added to the entire list, list 1.

5. Creating Jagged List

5.1. Jagged list example

The number of elements of the list inside the two-dimensional list may be different. Create two-dimensional jagged list as below.

```
1  list1 = [1, 2, 3, 4, 5]
2  list2 = []
3
4  for i in list1:
5      line = []
6      for j in range(i):
7          line.append(j)
8      list2.append(line)
9
10 print(list2)
```

```
[[0], [0, 1], [0, 1, 2], [0, 1, 2, 3], [0, 1, 2, 3, 4]]
```

Line 4, 5, 6

- In list 1, the horizontal size of the jagged list was stored in advance.
- The jagged list to be generated is list1. This list is initially empty.
- Repeat list 1 using for, take values from 1 to 5, and store them in i.
- Inside the for loop, repeat as the horizontal size i taken out of the for and add the elements using append. Then add an inner list, line, to list 2 in the outer loop.

5. Creating Jagged List

5.1. Jagged list example

- An example of a jagged list is as following. A jagged list is a two-dimensional list, and the number of items in the list inside the list is different.

- We can easily make a list of the following types.

```python
list2 = [[0],
         [0, 1],
         [0, 1, 2],
         [0, 1, 2, 3],
         [0, 1, 2, 3, 4]]

print(list2)
```

```
[[0], [0, 1], [0, 1, 2], [0, 1, 2, 3], [0, 1, 2, 3, 4]]
```

Like this, when the list size inside the list is different, it is called a jagged list.

5. Creating Jagged List

5.2. Jagged list using double for loop

It is possible to create a jagged list using a double for loop.

```
list1 = [1, 2, 3, 4, 5]
list2 = []
for i in list1:
    line = []
    for j in range(i):
        line.append(j)
        print(j, end = ' ')
    list2.append(line)
    print()
```

```
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
```

5. Creating Jagged List

5.3. Example of jagged list using double for loop and randint method

Inner values can be changed into random integers by using randint(1,100) of random module.

```python
import random

list1 = [1, 2, 3, 4, 5]
list2 = []
for i in list1:
    line = []
    for j in range(i):
        line.append(random.randint(1, 100))    #takes random value
    list2.append(line)

for i in list2:
    for j in i:
        print(j, end = ' ')
    print()
```

```
13
42 22
97 45 18
55 11 82 65
42 17 36 26 43
```

Paper coding

Try to fully understand the basic concept before moving on to the next step.

Lack of understanding basic concepts will increase your burden in learning this course, which may make you fail the course.

It may be difficult now, but for successful completion of this course we suggest you to fully understand the concept and move on to the next step.

Paper coding

Q1. Put the two-dimensional arrays [[10, 20], [30, 40], [50, 60] into the variable list_array and output 30. Do the correct indexing.

Conditions for Execution	30
Time	5 min

Write the entire code and the expected output results in the note.

Paper coding

Q2. Create a 2D list of 4 x 4 size with values ranging from 1 to 16 and print all the elements using the for loop.

Conditions For Execution	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Time	5 min

Write the entire code and the expected output results in the note.

Unit 13. Two-Dimensional Lists

Let's code

1. Generating Two-Dimensional List

1.1. Create 3x2 size two-dimensional list using loop statement

- Repeat for row size, 3 to create an empty list to use as an inner list and place the value in the list using append.
- Add the entire line list to list_array using an append.
- Make a list of elements in 3 rows and 2 columns using a loop statement.

```
list_array = []    # Create empty list

for i in range(3):
    line = []
    for j in range (2):
        line.append (0)
    list_array.append(line)

print(list_array)
```

```
[[0, 0], [0, 0], [0, 0]]
```

1. Generating Two-Dimensional List

1.2. Creating two-dimensional list using list comprehension expressions

- The code gets a little longer by using the for loop twice.
- By using a list compression expression, a two-dimensional list can be created with a single line of code.

```
list_array = [[0] * 2 for i in range(3)]
print(list_array)
```

```
[[0, 0], [0, 0], [0, 0]]
```

1. Generating Two-Dimensional List

1.2. Creating two-dimensional list using list comprehension expressions

- Python offers a function called list comprehension. Word comprehension in English means implication, inclusion, and intension, it is similar as what mathematicians define sets.
- Using the list comprehension expression, you can visit each item in the input list, perform a designated operation, and generate a list with an item value generated by the operation.

It becomes an element of a new list in a printed way.

For each element x in the input list,

Input list, tuple, String, etc.

```
[ i ** 2 for i in [1, 2, 3, 4, 5] ]
```

The comprehension expression allows you to create a list concisely and efficiently.

$$\{x^2 \mid x \in \mathbb{N}\}$$

```
[ 1, 4, 9, 16, 25 ]
```

1. Generating Two-Dimensional List

1.2. Creating two-dimensional list using list comprehension expressions

- Conditions may be added to the list implication using if. For example, if a set of even numbers among integers between 0 and 9 are expressed as a list connotation, it is as following.

Printed way

Input sequence

Conditional expression for individual input values.

```
[ i for i in range(10) if i % 2 == 0 ]
```

You can also filter individual input values using if conditional statement.

```
[ 0, 2, 4, 6, 8 ]
```

1. Generating Two-Dimensional List

1.3. Example code of creating two-dimensional list using list comprehension expression

```
1 list_array = [[0 for j in range(2)] for i in range (3)]
2 print(list_array)
```

```
[[0, 0], [0, 0], [0, 0]]
```

Line 1

- Repeat 0 twice using [0 for j range (2)] to change it into [0, 0], and repeat [0, 0] three times using for i in range (3) to make [[0, 0], [0, 0],[0, 0]].

2. Printing Two-Dimensional List Element

2.1. Printing two-dimensional list element using while statement

- It is also convenient to use a function len to find the size of the list when using while loop statements.
- Here, as in len(list_array), the row size of the 2D list is obtained, and then values in the inner list (column size) are assigned and printed.

```
1 list_array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
2 i = 0
3 while i < len(list_array):
4     a, b, c = list_array[i]
5     print(a, b, c)
6
7     i += 1
```

```
1 2 3
4 5 6
7 8 9
```

Line 2, 3

- The while statement is repeated for the number of element in list_array.
- If list_array[i] element of a, b, and c, that is i , is 0, 1, 2, and 3 will be assigned respectively.

3. Creating Jagged Lists

3.1. Creating and printing jagged lists

- Jagged list can also be simply created using comprehension expression.

```
list1 = [[0] * i for i in [1, 2, 3, 4, 5]]
print(list1)
```

```
[[0], [0, 0], [0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

- Code using double for loop to print the jagged list, list 1 created in the code above is as following.

```
for i in list1:
    for j in i:
        print(j, end = ' ')
    print()
```

```
0
0 0
0 0 0
0 0 0 0
0 0 0 0 0
```

3. Creating Jagged Lists

3.2. Append method

- Adding new item to the jagged list. Since list1[0] and list1[1] are also lists, 99 and 77 elements can be added using append method.

```
list1 = [[10 * i] * i for i in [1, 2, 3, 4, 5]]
print(list1)
```

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

```
list1[0].append (99)
list1[1].append(77)
print(list1)
```

```
[[10, 99], [20, 20, 77], [30, 30, 30], [40, 40, 40, 40], [50, 50, 50, 50, 50]]
```

4. Read and Save Data from a File

4.1. What is file?

- A logical unit used to store data in a computer's storage device is called a file.
- It is possible to store it in a storage device such as a hard disk or an external disk and call it again when necessary.

4. Read and Save Data from a File

4.2. Steps to use files in Python

Files can be read/write to the disk storage device.

1. Open file
   - Import files by specifying the location and file name in the storage device.
   - The file location in the computer storage device is called a path.
2. Use it depending on the purpose of use
   - It is possible to read, write, check content, and add new content to the file, and delete existing contents.
3. What does it mean to close the file?
   - Once the file has been used up, the allocated memory must be returned for use in the system in order to use the file.
   - Such return process is expressed in the term "close" of the file.

4. Read and Save Data from a File

4.3. Writing files and modes

- Function open opens and brings 'hello.txt' file.
  - Imports by specifying 'w' file open mode(write only)
- Open function returns a file object.
- File object f uses the write method to write text specified in a file.
- f.close method closes the file.

```
f = open('hello.txt', 'w')
f.write('hello world!')
f.close()
```

open('hello.txt', 'w') opens and brings the file in writing mode

4. Read and Save Data from a File

4.4. Open function

- It is a key function that deals with files in Python.
- It may have a file name and a factor of the file open mode.
- It is possible to designate the encoding format of a file through the encoding factor.
- If there is no designated path, the default path is where the Python script file is located.

4. Read and Save Data from a File

4.5. Various file input/output modes of the open function

Mode	Meaning
r	Short for Read, opens the file in read-only mode. This mode is the default mode for file opening. - If there is no file, an error occurs. - Since it is in read-only mode, an error occurs when trying to write a file.
w	Short for Write, opens the file in **write-only mode**. - If there is no file, a file is created, and if there is a file, it is overwritten (caution).
a	Short for Append, opens the file in write mode and adds the newly created contents after the existing file. - If there is no existing file, creates a new file and adds the contents.
x	This mode opens files exclusively for writing and is used to create new files. - Unlike a or w, an error occurs in this mode when there is a file. (Safety mode of w mode)
+	The + symbol is read/write mode and can be used as a combination of 'r+', 'w+', and 'r+t'. With 'r+', it can be opened and written in read mode. On the other hand, with 'w+', you can open it in write mode and read it.

4. Read and Save Data from a File

4.6. A mode that specifies whether to handle binary files or text files

Mode	Meaning
t	Short for text, opens or creates files in the form of text files. This mode is the default mode for file opening.
b	Short for binary, opens or creates files in binary file format.

4. Read and Save Data from a File

4.7. Open function example code

```
1 f = open('hello.txt', 'w') # Open file
```

```
1 f = open ('hello.txt', 'wt') # Open file in text
```

```
1 f = open('hello.txt', 'wb') # Open file in binary
```

```
1 f = open('hello.txt', 'r+t') # Text read/write mode
```

Line 1

- 'w' : returns a file object by reading the file in write mode (w). This file object f can write text in a specified file by calling the write method.
- 'wt': returns a file object by reading the file in write mode (w). This file can call the write method to write text to a specified file. Here, you can open and write a file in text (t) mode.
- Since the text mode is a default mode, it can be omitted.
- 'wb': wb mode is a binary mode and you can write files.
- 'r+t': The default mode is a read mode, but you can also write files. In this mode, an error occurs if there is no file.

4. Read and Save Data from a File

4.8. open, write function

```
1 f = open('hello. txt', 'w+t') # Text write/read mode
```

```
1 f = open('hello.txt','a+t') #Add text
```

```
1 f.write('Hello World!')    # Write in hello. txt file
```

12

```
1 f.write('Love of my life, you\'ve hurt me\n')
2 f.write('You\'ve broken my heart and now you leave me\n')
3 f.write ('Love of my life, can\'t you see?\n')
```

32

Line 1

- 'w+t': The default mode is a write mode, but you can also read files.
- 'a+t': Adds content at the end of the existing file, leaving all the contents of the existing file.
- It can be read and created if there is no file.

4. Read and Save Data from a File

4.8. open, write function

```
1 f = open('hello. txt', 'w+t') # Text write/read mode
```

```
1 f = open('hello.txt','a+t') #Add text
```

```
1 f.write('Hello World!')    # Write in hello. txt file
```

12

```
1 f.write('Love of my life, you\'ve hurt me\n')
2 f.write('You\'ve broken my heart and now you leave me\n')
3 f.write ('Love of my life, can\'t you see?\n')
```

32

Line 1-3

- Write a text file using the f.write function. String input is possible.
- Here, you need to write the escape string \n to break the line in the file.
- Quotations cannot be used inside '...' indicating the beginning and end of a string. Therefore, \' can be used to print quotes.

4. Read and Save Data from a File

4.9. Close method

Focus Reasons for using close method

- Data is generally buffered and then processed in the processor responsible for input/output of the computer.
- The buffer used here is a temporary memory space.
- If a file is uploaded into the buffer, which is a temporary memory space, the system's memory will continue to be used.
- To write files efficiently, it is used to collect data to a certain size and read or write them all at once.
- If the buffer contents are sent to the disk through the close method, the memory as much as the buffer size becomes empty.
- Therefore, memory efficiency increases only when the file is closed using the close method.

```
f.close() # Close file
```

One More Step

Let's look at the need for buffers and the close method.

- Reading and writing data through a computer's hard disk is tens to hundreds of times slower than reading and writing data within a central processing unit or memory. Therefore, it is faster to collect and process reading and writing data to disk at once as much as possible. For this reason, it is faster to store 1000 bytes of data in the computer's memory 10 times in units of 100 bytes than to store 1 byte each on disk 1000 times.
- Therefore, it is generally more effective to use a buffer to collect data from the write command in units of 100 bytes and write it at once when it becomes 100 bytes. What the close method does is to store information in the buffer on disk by giving information that no more write commands are to be sent to this file.

4. Read and Save Data from a File

4.10. Read file

- Read the file using the read method.
- All strings (full content) in the file are returned as one string regardless of line breaks.

```
f = open ('hello.txt', 'r')  # Open file
s = f.read()                 # Read hello.txt file
print(s)                     # print content of the file
f.close ()                   # close file
```

```
Hello World!Love of my life, you've hurt me
You've broken my heart and now you leave me
Love of my life, can't you see?
```

open('hello.txt', 'r') opens and brings the file in read mode.

4. Read and Save Data from a File

4.10. Read file

- If 5 is given as a factor of the f.read method, it reads 5 characters from the file and return them in string.
- Only five characters of 'Hello' are printed.

```
1 f = open ('hello.txt', 'r')  # Open file
2 s = f.read(5)                # Read five letters in hello.txt file
3 print(s)                     # print content of the file
4 f.close ()                   # close file
```

```
Hello
```

Line 1, 2

- Open the file in read mode.
- If you add 5 using the read method, only five characters in the file are read.

4. Read and Save Data from a File

4.11. Readline method

Read two lines of the hello.txt file using the readline method and print them.

```
1 f = open('hello.txt', 'r') # Open file
2 s = f.readline()           # Read first line of the file 'AAA'
3 print(s, end = '')         # print the line
4 s = f.readline()           # Read second line of the file 'BBB'
5 print(s, end = '')         # print the line
6 f.close()                  # Close file
```

```
Hello World!Love of my life, you've hurt me
You've broken my heart and now you leave me
```

Line 2, 4

- The readline method reads the first line of the file.
- If you write it one more time, it reads the next line.

4. Read and Save Data from a File

4.12. Add files

- Use 'a+' mode
  - Add new contents to the back of a file that has already been created.
- a is append mode
- + is a mode to create a new file if there is no existing file.

4. Read and Save Data from a File

4.12. Add files

```
1 f = open('hello.txt', 'a+') # Open file
2 f.write('This will be appended.\n')
3 f.write('This too.\n')
4 f.close()
```

```
1 f = open ('hello.txt', 'r')  # Open file
2 s = f.read()                 # Read hello.txt file
3 print(s)                     # print content of the file
4 f.close ()
```

```
Hello World!Love of my life, you've hurt me
You've broken my heart and now you leave me
Love of my life, can't you see?
This will be appended.
This too.
```

Words added

Line 2, 3

- Add line-breaking character (\n) and strings using write method.

4. Read and Save Data from a File

4.13. Example of file reading and writing

- Let's find available applications.
- Receive five integers from users and store them in data 5.txt.
- Read data5.txt and print the sum and mean of the integers.

```
f = open ('data5.txt', 'w')            # Open file in writing mode
for _ in range (5):
    n = input('Enter integer: ')
    f.write(n)                         # Write the input in string
    f.write('\n')                      # Write the value and change line
f.close()                              # close file
```

```
Enter integer: 10
Enter integer: 20
Enter integer: 30
Enter integer: 40
Enter integer: 50
```

4. Read and Save Data from a File

4.13. Example of file reading and writing

When you enter 10, 20, 30, 40, and 50, and then look at the Jupiter homepage with the file executed by Jupiter, the data5.txt file is created as following.

4. Read and Save Data from a File

4.13. Example of file reading and writing

- Read integers from the file and print the sum and mean of those integers.
- Since 10 stored in the file is a string "10", addition operations are possible only when it is replaced with an integer using int.

```
1  f = open('data5.txt', 'r')                                    # open file in reading mode
2  su = 0
3  for _ in range(5):
4      n = int(f.readline())                                     # read a number within a line of data5.txt file
5      su += n                                                   # accumulate and add to su.
6
7  print('Sum of the numbers = {}, average = {}'.format(su, su/5))   # print sum and average
8  f.close()                                                     # close file
```

```
Sum of the numbers = 150, average = 30.0
```

Line 4, 5

- n converts the letters read through the readline into int type.
- Add the n value to the su variable by accumulating it.

4. Read and Save Data from a File

4.14. With statement

- with ~ as : statement can be used to simply write and close text in a file.

```
with open ('hello.txt', 'w') as f:        # automatically open and close file
    f.write('Hello World!')               # write in hello.txt file
```

- There is no need to use the finally clause because f.close is automatically performed after file writing is completed.

One More Step

With statement in Python

- The with statement is a way to clarify grammar and handle exceptions easily.
- It defines the __enter__ and __exit_ functions executed by the context manager, replacing the code executed at the front and back of the with syntax body.
- Using the with statement, concise code is possible on behalf of try-except-finally.
- Since the f.write operation always contains the possibility of errors, it must be coded as follows using the try clause.
- It is also important to close the file, so you must put this code after the final clause.
- Such cumbersome work can be done simply using with statement.

```
f = open('hello.txt', 'w')          # open file
try:
    f.write('Hello World!')         # write in hello.txt file
finally:
    f.close()                       # close file
```

One More Step

What is context manager?

- The context manager is a Python class that plays a role in accurately allocating and providing resources at the desired timing.
- Computer works using memory and CPU, and these computing elements are called resources.
- If this resource is not properly managed, the computer will not work.
- The main class that manages these resources is Python's context manager.
- Closing after opening a file is the main role of the resource management program, so the context manager plays this role in Python.
- The try-except-finally on the previous page is described in Unit 19 as an exception handling statement.
- Details on these are beyond the scope of this course.

One More Step

- With statement is also useful for simplifying the syntax that accesses the page by opening a web page as follows.
- To open a web page, use a package called urlopen. You can import the contents of a web page through this package. In this case, the with statement is useful as well.
- The following code is a code that brings the content of a web page called python.org.

Python Context Manager returns HTTPResponse object and calls a close method that automatically terminates Internet access after this section. (Otherwise, this Python program continues to access the Internet and wastes memory.)

```python
import urllib.request

with urllib.request.urlopen('http://python.org/') as response:
    html = response.read()
    print(html)
```

```
b'<!doctype html>\n<!--[if lt IE 7]>   <html class="no-js ie6 lt-ie7 lt-ie8 lt-ie9">   <![endif]-->\n<!--[if IE 7]>
<html class="no-js ie7 lt-ie8 lt-ie9">          <![endif]-->\n<!--[if IE 8]>      <html class="no-js ie8 lt-ie9">
<![endif]-->\n<!--[if gt IE 8]><!--><html class="no-js" lang="en" dir="ltr">  <!--<![endif]-->\n\n<head>\n    <meta char
set="utf-8">\n    <meta http-equiv="X-UA-Compatible" content="IE=edge">\n\n    <link rel="prefetch" href="//ajax.googlea
pis.com/ajax/libs/jquery/1.8.2/jquery.min.js">\n    <link rel="prefetch" href="//ajax.googleapis.com/ajax/libs/jqueryui/
1.12.1/jquery-ui.min.js">\n\n    <meta name="application-name" content="Python.org">\n    <meta name="msapplication-tool
tip" content="The official home of the Python Programming Language">\n    <meta name="apple-mobile-web-app-title" conten
t="Python.org">\n    <meta name="apple-mobile-web-app-capable" content="yes">\n    <meta name="apple-mobile-web-app-stat
us-bar-style" content="black">\n\n    <meta name="viewport" content="width=device-width, initial-scale=1.0">\n    <meta
```

Pair programming

Pair Programming Practice

Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.

Pair Programming Practice

Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

Collaborative learning environment

A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor's question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students "divide and conquer." When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

Pair programming

Q1. Write a program that generates a multidimensional array of n×n size, based on the number of inputs, by receiving two or more n as inputs from users. In this case, the content of the arrangement should be displayed so that the values of 0 and 1 intersect in a checkered pattern.

Output example

```
Enter n: 5
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
```

Unit 14.

Dictionary Method 1

Unit learning objective (1/3)

Learning objectives

- ✓ Be able to add or modify new values using the dictionary key.
- ✓ Be able to find and delete any item inside the dictionary.
- ✓ Be able to delete all items inside the dictionary using the clear method of the dictionary.
- ✓ Be able to extract a key-value item using a method referring to a key in the dictionary.

Unit learning objective (2/3)

Learning overview

- ✓ Add and modify the dictionary key-value using the method of adding and modifying it.
- ✓ Delete the arbitrary value using the method of deleting the dictionary's random key-value.
- ✓ Delete all keys using the method that delete all dictionary's key-value.
- ✓ Save the dictionary key-value to another variable using the method that brings the specific key-value of the dictionary.

Concepts you will need to know from previous units

- ✓ Understand that dictionaries are accessed with keys rather than with indexes.
- ✓ Able to access and assign value to the dictionary key and perform dictionary operation.
- ✓ Understand the difference between list and dictionary and be able to declare and utilize the appropriate data type (list, dictionary) according to the situation.

Unit learning objective (3/3)

Keywords

- Dictionary
- Key-Value
- Add, Edit
- Delete
- Save

Unit 14. Dictionary Method 1

Mission

1. Real world problem

1.1. Difficulties for self-employed people of cafes and restaurants due to rising raw material prices and labor costs

- Local commercial districts centered on self-employed people such as cafes and restaurants account for a large proportion of local residents' economic activities.
- However, the burden of continuous rising raw material prices and labor costs may cause management difficulties for self-employed people.
- A lot of effort is needed to minimize operating costs, and we will develop programs to help these operations.
- Through the previous mission, we learned how to print out the cafe menu. This mission will improve the previous function to implement the function of adding new menus and prices.

1. Real world problem

1.2. Program that tells you the price when you call out the menu

- When hiring new employees at a cafe, it takes a lot of time and money to train them to properly acquire menus and prices and respond according to the manual.
- Menu and price often change depending on the situation. For example, some menus are often added or deleted seasonally and quarterly.
- Therefore, we are going to develop a program for staffs of a cafe. This program is a program that informs customers of the price as soon as they call the menu.
- We want to put data so that the menu and price are linked and the price can come out immediately if the menu is called.
- Let's create a cafe menu program that functions like above.

1. Real world problem

1.3. The etymology of café

- Place and location of drinking coffee, tea, etc. In French, coffee is called a cafe, which has hardened into a "house selling coffee."
- Today's cafes are gaining popularity as places to enjoy reading or use smartphones or laptop computers using the wireless Internet.
- Global cafe chain operators such as Starbucks are steadily enjoying popularity in countries such as Korea and Japan with their advanced strategies and new user experiences.
- Recently, Starbucks is providing services suitable for the mobile era such as SIREN Order and is leading innovation in ordering methods.

2. Solution

2.1. Café menu program

- In this mission, we will enter the menu in the cafe and print out the price of the corresponding menu. Furthermore, we will implement the function of entering new menus and prices.
- These commands will work by typing the commands into the Command Prompt.
- A command prompt '$' appears in the execution window, and the input command is expressed as '<' and the search command is expressed as '>'.
- Input is menu: price form. Enter the menu to search them.
- If you want to exit the cafe menu program, enter q.

3. Mission

3.1. This is how café menu program works.

```python
print("Start café menu program... Press q to exit")
cafe_menu = {'Ice coffee' : 3000}

while True:
    str = input('$ ')
    if str.startswith('q'):
        break;
    command = str[0]

    if command == '<':
        str = str.replace('<', "")
        inputStr = str.split(':')
        if len(inputStr) < 2:
            print('input error')
            continue
        else:
            cafe_menu[inputStr[0].strip()] = inputStr[1].strip()

    elif command == '>':
        str = str. replace('>',"")
        inputStr = str.strip()
        if inputStr in cafe_menu:
            print(cafe_menu[inputStr])
        else:
            print('{} not in the menu.'.format(inputStr))
    elif command == 'q':
        break
    else:
        print('input error.')
print("exiting cafe menu program.")
```

3. Mission

3.1. This is how café menu program works.

```
Start café menu program... Press q to exit
$ > Ice coffee
3000
$ > green tea
green tea not in the menu.
$ < espresso : 5000
$ espresso
input error.
$ > espresso
5000
$ q
exiting cafe menu program.
```

3. Mission

3.2. Programming plan

Pseudocode

[1] Start

[2] Reset café menu dictionary.

[3] while True

[4] Input command and café menu.

[5] if the first command is '<'(Input), then
[6] Get menu and price from the string input
Add menu and price to café dictionary.
[7] else if first command'>'(output), then
[8] Print the menu.

[9] else if the first command is 'q',

[10] exit the program [13] End

[11] else print error

[12] Exit program

[13] End

Flowchart

3. Mission

3.3. Café menu program final code

```python
print("Start café menu program... Press q to exit")
cafe_menu = {'Ice coffee' : 3000}

while True:
    str = input('$ ')
    if str.startswith('q'):
        break;
    command = str[0]

    if command == '<':
        str = str.replace('<', "")
        inputStr = str.split(':')
        if len(inputStr) < 2:
            print('input error')
            continue
        else:
            cafe_menu[inputStr[0].strip()] = inputStr[1].strip()

    elif command == '>':
        str = str. replace('>',"")
        inputStr = str.strip()
        if inputStr in cafe_menu:
            print(cafe_menu[inputStr])
        else:
            print('{} not in the menu.'.format(inputStr))
    elif command == 'q':
        break
    else:
        print('input error.')
print("exiting cafe menu program.")
```

Unit 14. Dictionary Method 1

Key concept

1. Add and Edit Key-Value Together

1.1. Add, edit dictionary data type

- It is possible to add and delete information to the dictionary as necessary.
- One of the important functions to do this is to add a key-value pair.
- There is a method of adding a key-value in the same format as dic[key] = value.
- However, there are cases where the default value needs to be added or updated as necessary.
- Let's take a closer look at the advanced methods of dictionary, setdefault and update.
  - setdefault: adds default key-value pairs
  - update: modifies the value of a key and adds a key-value pair if there is no key

1. Add and Edit Key-Value Together

1.2. Dictionary code without setdefault

- The code below is a code that counts the number of alphabetic letters in the given list s and stores them in advance.
- a appears twice, b appears three times, and c appears twice. Let's put this information in the dictionary and print it out.

```
1 str1 = ['a', 'b', 'c', 'b', 'a', 'b', 'c']
2 dic = {}
3 for ch in str1:
4     if ch not in dic.keys():
5         dic[ch] = 0
6     dic[ch] += 1
7
8 print('alphabet counting :', dic)
```

```
alphabet counting : {'a': 2, 'b': 3, 'c': 2}
```

Line 4, 5, 6

- Examine whether the value of ch is in the key of dic or not.
- If not, add a key and initialize the value to zero.
- Add 1 to the key of dic.

1. Add and Edit Key-Value Together

1.2. Dictionary code without setdefault

- In order to construct a dictionary, it is convenient to set the default values for each key.
- However, the dictionary used here is an empty dictionary. Therefore, the default value does not exist.
- To use this dictionary, it is necessary to check whether the alphabet ch is included in the dictionary key in the if condition and then initialize it as shown in dic[i] = 0, if not.
- Let's look at the setdefault method that makes the functionality of the dictionary more convenient.

The dic of this code stores the frequency of appearance of each alphabet.

1. Add and Edit Key-Value Together

1.3. Dictionary code with setdefault

- Python's dictionary data structure provides a setdefault method to avoid if conditional clauses of the previous code.
- If the key value is passed as the first factor and the default value as the second factor as follows, the default value for the key is specified as 0.

```
1 str1 = ['a', 'b', 'c', 'b', 'a', 'b', 'c']
2 dic = {}
3 for ch in str1:
4     dic.setdefault(ch, 0)
5     dic[ch] += 1
6     print(list(dic.items()))
7
8 print('alphabet counting :', dic)
```

```
[('a', 1)]
[('a', 1), ('b', 1)]
[('a', 1), ('b', 1), ('c', 1)]
[('a', 1), ('b', 2), ('c', 1)]
[('a', 2), ('b', 2), ('c', 1)]
[('a', 2), ('b', 3), ('c', 1)]
[('a', 2), ('b', 3), ('c', 2)]
alphabet counting : {'a': 2, 'b': 3, 'c': 2}
```

Line 4, 5, 6

- The setdefault returns the key if it has a key and returns a second factor if it does not.

1. Add and Edit Key-Value Together

1.3. Dictionary code with setdefault

- Python's dictionary data structure provides a setdefault method to avoid if conditional clauses of the previous code.
- If the key value is passed as the first factor and the default value as the second factor as follows, the default value for the key is specified as 0.

```
1 str1 = ['a', 'b', 'c', 'b', 'a', 'b', 'c']
2 dic = {}
3 for ch in str1:
4     dic.setdefault(ch, 0)
5     dic[ch] += 1
6     print(list(dic.items()))
7
8 print('alphabet counting :', dic)
```

```
[('a', 1)]
[('a', 1), ('b', 1)]
[('a', 1), ('b', 1), ('c', 1)]
[('a', 1), ('b', 2), ('c', 1)]
[('a', 2), ('b', 2), ('c', 1)]
[('a', 2), ('b', 3), ('c', 1)]
[('a', 2), ('b', 3), ('c', 2)]
alphabet counting : {'a': 2, 'b': 3, 'c': 2}
```

Line 4, 5, 6

- Therefore, it is possible to see the effect that all appearances of alphabetic characters are initialized to zero through dic.setdefault(ch, 0).

1. Add and Edit Key-Value Together

1.3. Dictionary code with setdefault

- Python's dictionary data structure provides a setdefault method to avoid if conditional clauses of the previous code.
- If the key value is passed as the first factor and the default value as the second factor as follows, the default value for the key is specified as 0.

```
1 str1 = ['a', 'b', 'c', 'b', 'a', 'b', 'c']
2 dic = {}
3 for ch in str1:
4     dic.setdefault(ch, 0)
5     dic[ch] += 1
6     print(list(dic.items()))
7
8 print('alphabet counting :', dic)
```

```
[('a', 1)]
[('a', 1), ('b', 1)]
[('a', 1), ('b', 1), ('c', 1)]
[('a', 1), ('b', 2), ('c', 1)]
[('a', 2), ('b', 2), ('c', 1)]
[('a', 2), ('b', 3), ('c', 1)]
[('a', 2), ('b', 3), ('c', 2)]
alphabet counting : {'a': 2, 'b': 3, 'c': 2}
```

Line 4, 5, 6

- Key value changes every time when an alphabet is entered through the print statement inside the code.

1. Add and Edit Key-Value Together

1.3. Dictionary code with setdefault

- If a key and default values are specified, such as setdefault(key, value), the default values are stored in the values and the values are returned.
- The following example is a code that adds a key 'f' to a dictionary called dic, stores 50 in a default value, and print it.

```
1 dic = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
2 dic.setdefault('f', 50)
3 print(dic)
```

```
{'a': 10, 'b': 20, 'c': 30, 'd': 40, 'f': 50}
```

Line 2

- Since there is no key 'f', add it and put the value as 50.

The default value for the 'f' key of this dic is 50.

1. Add and Edit Key-Value Together

1.4. Update changes value of key.

- The update(key=value) method modifies the value of the key in the dictionary.
- For example, if the dictionary is dic = {'a' : 10} (where the key is a string), as shown in x.update (a = 90), key names and values can be specified except single or double quotes from the key.
- In this case, the value of the key 'a' may be modified to 90 as following.

```
1 dic = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
2 dic.update(a = 90)
3 print (dic)
```

```
{'a': 90, 'b': 20, 'c': 30, 'd': 40}
```

Line 2

- Change a to 90.

1. Add and Edit Key-Value Together

1.5. When update method adds key that does not exist.

- If the key does not exist in the dictionary, the update method also adds a key-value pair.
- As shown below, there is no key 'e' in the dic, so when dic.update (e=50) is executed, you can see that 'e':50 is added.

```
dic.update(e = 50)
print (dic)
```

```
{'a': 90, 'b': 20, 'c': 30, 'd': 40, 'e': 50}
```

1. Add and Edit Key-Value Together

1.6. Modify multiple items with update method

- Update method can modify values at once with multiple key-values separated by commas.
- Same here, if a key exists, modify the value of the corresponding key, and if not, add a key-value pair.
- Next, modify the value of the key 'a' to 900, and add 'f': 60.

```
dic.update(a = 900, f = 60)
print(dic)
```

```
{'a': 900, 'b': 20, 'c': 30, 'd': 40, 'e': 50, 'f': 60}
```

1. Add and Edit Key-Value Together

1.7. Precautions for using the update method

Focus The form of update (key=value) can only be used when the key is a string.

If the key is a number, the value may be modified by inserting a dictionary like update(dictionary).

```
dic2 = {1: 'one', 2: 'two'}
dic2.update({1: 'ONE', 3: 'THREE'})
print(dic2)
```

```
{1: 'ONE', 2: 'two', 3: 'THREE'}
```

1. Add and Edit Key-Value Together

1.8. How to modify the update method key when it's not a string

- Another way to modify the key-value is to use a list and a tuple.
- Update (list) and update (tuple) modify the values with lists and tuples.
- Here, the list creates a list with keys and values in the form of [Key1, Value1] and [Key2, Value2]], and lists key-value pairs by putting this list back into the list.
- Tuple is also in the same format as above.

List

```
dic2.update([[2, 'TWO'], [4, 'FOUR']])
print(dic2)
```

```
{1: 'ONE', 2: 'TWO', 3: 'THREE', 4: 'FOUR'}
```

Tuple

```
dic2.update(((2, 'TwO'), (4, 'FOUR')))
print(dic2)
```

```
{1: 'ONE', 2: 'TwO', 3: 'THREE', 4: 'FOUR'}
```

1. Add and Edit Key-Value Together

1.9. Difference between setdefault and update

- Setdefault can only add key-value pairs, and the values of already existing keys cannot be modified.
- However, update can both add and modify key-value pair.
- Even if the key ‘a’ already included is stored as 90 using setdefault, the value of 'a' does not change.

```
dic = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
print(dic)
dic.setdefault('a', 90)
print(dic)
```

```
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
```

2. Delete Specific, Random Key-Value

2.1. Pop method

The pop method deletes a specific key-value pair and returns it.

```
dic ={'a': 10, 'b': 20, 'C': 30, 'd': 40}
print(dic.pop('a'))
print(dic)
```

```
10
{'b': 20, 'C': 30, 'd': 40}
```

- Pop(key) returns the deleted value after deleting a specific key-value pair from the dictionary.
- Next, delete the key 'a' from the dictionary dic and return 10.

2. Delete Specific, Random Key-Value

2.2. Pop application

- If you specify a default value like pop(key, default value), it deletes the key-value pair and returns the deleted value when there is a key in the dictionary, but it only returns the default value when there is no key.
- Since there is no key 'f' in dictionary dic, 8 designated as the default value is returned.

```
1 dic ={'a': 10, 'b': 20, 'C': 30, 'd': 40}
2 print(dic.pop('a'))
3 print(dic)
```

```
10
{'b': 20, 'C': 30, 'd': 40}
```

```
1 print(dic.pop('f', 8))
```

```
8
```

Line 1

- Since there is no key with the letter 'f' in the dic, 8 is returned.

2. Delete Specific, Random Key-Value

2.3. Popitem method

Popitem deletes any key-value pair and returns the pair.

```
dic = {'a': 10, 'b': 20, 'C': 30, 'd': 40}
print (dic.popitem())
```

```
('d', 40)
```

```
print(dic)
```

```
{'a': 10, 'b': 20, 'C': 30}
```

- Popitem deletes any key-value pair from the dictionary and returns the deleted key-value pair in tuple.
- The behavior of this method depends on the Python version, which deletes the last key-value pair in Python 3.6 and above and any key-value pair in 3.5 and below.
- This lecture material is based on Python version 3.6 or higher, so the last element is returned.

3. Delete All Key-Value : clear

3.1. clear method

Clear deletes all the key-value pairs in dictionary.

```
dic = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
print(dic)
dic.clear()
print(dic)
```

```
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
{}
```

- After clear, dictionary dic becomes an empty dictionary {}.

4. Bring a value of a specific key and save it : get

4.1. get method

Get(key) takes the value of a specific key from the dictionary.

Ex Following brings the value of key 'a' from dictionary dic

```
dic = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
a = dic.get('a')
print(a)
```

```
10
```

- If you specify a default value like get(key default value), it returns the value of the key when the dictionary has it but returns the default value when there is no key.
- Since there is no key 'z' in dictionary dic, it returns 0 that is specified as the default value.

```
b = dic.get('z', 0)
print(b)
```

```
0
```

5. Items, Keys, Values

5.1. Print key value

- The most important operation in dictionary is to find the associated value with the key as following.

```
dic = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
dic['a']
```

```
10
```

- To output all keys used in the dictionary, use the method keys as following.

```
dic = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
dic.keys()
```

```
dict_keys(['a', 'b', 'c', 'd'])
```

5. Items, Keys, Values

5.2. Print value

- Values returns all the values used in the dictionary.
  - However, values is used to return all values used in the dictionary.

```
dic.values()
```

```
dict_values([10, 20, 30, 40])
```

5. Items, Keys, Values

5.3. Print both key and value

- Items prints both key and value.
  - In addition, items may be used to return all values inside the dictionary.
  - When a function such as dic.items is called in the for loop, the (key,value) tuple is returned, so it can be used as following.

```
1 dic.items()
```

```
dict_items([('a', 10), ('b', 20), ('c', 30), ('d', 40)])
```

```
1 for str1, num in dic.items():
2     print(str1, ':', num)
```

```
a : 10
b : 20
c : 30
d : 40
```

Line 1

- Values corresponding to keys and values of the dictionary are assigned to str1 and num, respectively.

5. Items, Keys, Values

5.4. Various methods of dictionary and the functions

Method	Function
keys()	Returns all keys in the dictionary.
values()	Returns all values in the dictionary.
items()	Returns all items in the dictionary in [key]:[value] pairs.
get(key)	Returns the value of the key. If there is no key, return None.
pop(key)	Returns the value of the key, and deletes the item. If there is no key, it causes a KeyError exception.
popitem()	Returns the randomly selected item and deletes the item.
clear()	Delete all items in the dictionary.

Unit 14. Dictionary Method 1

Paper coding

Try to fully understand the basic concept before moving on to the next step.

Lack of understanding basic concepts will increase your burden in learning this course, which may make you fail the course.

It may be difficult now, but for successful completion of this course we suggest you to fully understand the concept and move on to the next step.

Paper coding

Q1. Let's create a dictionary named person_dic with the following contact information on your phone. Print this information using the for loop to show the output results below.

Conditions for Execution	`'Last Name': 'Doe', 'First Name': 'David', 'Company': 'Samsung'`
Time	5 min

Last Name
First Name
Company

```
Last Name  : Doe
First Name  : David
Company  : Samsung
```

Write the entire code and the expected output results in the note.

Paper coding

Q2. Let's write a program that performs inventory management at a convenience store. To this end, inventory of items sold at convenience stores is stored in the items dictionary as shown in the example below. Write a program that receives the name of the item from users and returns the inventory of the item. Suppose that it is a very small convenience store and the items treated are as following.

Example program execution results.	`Enter name of the item: Milk` `1`
Time	5 min

Items

```
items = {"Coffee": 7, "Pen":3, "Paper cup": 2, "Milk": 1, "Coke": 4, "Book":5}
```

Write the entire code and the expected output results in the note.

Unit 14. Dictionary Method 1

Let's code

1. Let's Do Coding that Processes Text Data

1.1. Text data

- Text has long been used by humans as the most important mean of efficiently exchanging information.
- Text data may be divided into structured documents (HTML, XML, CSV, JSON files) and unstructured documents (text in natural language).
- In general, since source data is not in a processed form, it needs to be modified into complete data.

1. Let’s Do Coding that Processes Text Data

1.2. Text data and word cloud expression

Train
Perform
Computer
Task
Time
Algorithm
Predict
Same
Theory
Such
Two
Field
Output
Set
Learn
Statistics
Method
Data
Supervised
Approach
Cluster
Model
Machine
Example
Use
Label
More
Function
Goal
One
Call
Input
Type
Find
Program
Based
Article
Game
Problem
Differ
Main
Many
Out
Loss
Fit
Self
Step
New
Unsupervised
Known
Term
Similar
Mining
Research
Pattern
General
Each
Teach
Being
Network
Work
Between
Represented
Knowledge
Classification
Digits
Certain
System
Complex
Optimal
Discipline
Neural
Reinforcement
Answers
Artificial
Analysis
Regression
Related
Separate
Measure
Learner
Function
Assign
Divided
Needed
Execute
Sample
Employ
Discovery
Science
Within
Create
Build
Respect
Teacher
Play
Less
Local
Intelligence
Feature
Without
Space
Turn
Involves
Much
Linear
Cases
Provided
Bounds
Sample
Following
Hypothesis
Human
Pre
Email
Study
Toward
Part
Rule
Density
Vector
Value
Dr
Recognition
Error
Cs
Any
Learning
Data
Set
Input
Output
Model
Machine
Train
Time
Task
Problem
Function
Goal
Use
Label
Term
Example
Pattern
Cluster
Approach
Field
Method
Statistics
Supervised
Unsupervised
Reinforcement
Research
Artificial
Neural
Mining
Divided
Optimal
Learner
Discipline
Intelligence
Feature
Without
Space
Turn
Involves
Much
Linear
Cases
Provided
Bounds
Sample
Following

1. Let's Do Coding that Processes Text Data

1.3. Split method

- Split method cuts strings.
  - Suppose that the words in the string are separated by separators such as commas and blanks.
  - For example, if the method split is used for string s as below, one string can be separated into three strings using a space as a separator.

```python
s = 'Welcome to Python'
s.split()
```

```
['Welcome', 'to', 'Python']
```

1. Let's Do Coding that Processes Text Data

1.4. Example of using split

- On the other hand, if there are years, months, and days divided by periods as below, if period (.) is given as a factor in the split method, characters can be classified by period.

```
s = '2021.8.15'
s.split('.')
```

```
['2021', '8', '15']
```

- Separate 'Hello, World!' with a comma as shown below.

```
s = 'Hello, World!'
s.split(",")
```

```
['Hello', ' World!']
```

- The execution results show that it was separated into 'Hello' and 'World!'. However, it looks awkward because there is a space in front of W.
- These spaces can be removed by the strip method.

1. Let's Do Coding that Processes Text Data

1.5. Strip method

- Strip removes the space before and after the string.
  - However, this method cannot be applied if there are more than two spaces or spaces before and after the string.
  - The code below is not properly separated because an incorrect separator character is used.

```
s = 'Hello, World!'
s.split(',  ')
```

```
['Hello, World!']
```

1. Let's Do Coding that Processes Text Data

1.5. Strip method

- The strip removes the space before and after the string.
  - If there is a space after a comma as below, it is recommended to delete the space after dividing it by commas.
  - It would be desirable to remove the space before and after the string using the strip method as below.
  - The following code is as an abbreviation of the list and will be covered in detail in Unit 19.
  - This expression method often appears in Python coding.

```
s = 'Welcome, to,  Python, and ,  bla, bla   '
s.split(',')   # separate s using comma as separator
```

```
['Welcome', ' to', '  Python', ' and ', '  bla', ' bla   ']
```

```
s = 'Welcome, to,  Python, and ,  bla, bla   '
[x.strip() for x in s.split(',')]   # delete space before and after separator character.
```

```
['Welcome', 'to', 'Python', 'and', 'bla', 'bla']
```

Let's code

- The strip removes the space before and after the string.
  - When processing string data, the strip method is responsible for removing unwanted spaces from the string.
  - The lstrip removes the left space of the string.
  - rstrip removes the space on the right side of the string.

```
'   Hello, World!   '      '   Hello, World!   '      '   Hello, World!   '
          ↓                          ↓                          ↓
       strip                      lstrip                     rstrip
          ↓                          ↓                          ↓
'Hello, World!'            'Hello, World!   '          '   Hello, World!'
```

```python
s = "   Hello, World!   "
s.strip()
```

```
'Hello, World!'
```

```python
s.lstrip()
```

```
'Hello, World!   '
```

```python
s.rstrip()
```

```
'   Hello, World!'
```

1. Let's Do Coding that Processes Text Data

1.6. Strip method and how to delete specific characters

- If you want to delete a particular character before and after the string, transfer the character as a factor of strip.
- The code below is to delete the '#' character.

```
s = "#######this is an example#####"
s.strip('#')
```

```
'this is an example'
```

- rstrip and lstrip can also include certain characters as factors.

```
s = "#######this is an example#####"
s.lstrip('#')
```

```
'this is an example#####'
```

```
s.rstrip('#')
```

```
'#######this is an example'
```

1. Let's Do Coding that Processes Text Data

1.7. Join method that connects strings

- If split is a function of separating a string into a substrings, join is a function that collects the substring and makes it a string. When calling join, you can designate characters that act as adhesives.

```
1 ','.join(['apple', 'grape', 'banana'])
```

Connects three words using comma

```
'apple,grape,banana'
```

```
1 '-'.join('010.1234.5678'.split('.'))  # change phone number with . to -
```

```
'010-1234-5678'
```

- From the example above, we can see that join puts adhesive characters only between strings and not before or after strings.

1. Let's Do Coding that Processes Text Data

1.7. Join method that connects strings

- If the join string is not specified, it returns the string without spaces.
  - If the string to be connected is not specified, the string is attached and returned as below.

```
''.join(['apple', 'grape', 'banana'])
```

```
'applegrapebanana'
```

1. Let's Do Coding that Processes Text Data

1.7. Join method that connects strings

- Join is sometimes used to collect separated characters.
  - join is also used to collect the characters separated by the list function as follows and turn them back into the original string.

Ex Let's connect clist, "hello world" that is changed from string to list, using join.

```
s = 'hello world'
clist = list(s)
clist
```

```
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
```

```
''.join(clist)
```

```
'hello world'
```

1. Let's Do Coding that Processes Text Data

1.8. Using split and join together

- By using split and join together, we can remove unnecessary spaces from the string.

```
a_string = 'Actions \n\t speak louder than words'
a_string
```

```
'Actions \n\t speak louder than words'
```

```
print(a_string)
```

```
Actions
	 speak louder than words
```

```
word_list = a_string.split()
word_list
```

```
['Actions', 'speak', 'louder', 'than', 'words']
```

```
refined_string = " ".join(word_list)    # line change and tap disappears from a_string
print(refined_string)
```

```
Actions speak louder than words
```

- After removing the space using split, the string is attached using join.

1. Let's Do Coding that Processes Text Data

1.9. Capital letter and lower-case letter conversion

- In a string, the function of changing uppercase letters to lowercase letters is lower, and the opposite method of changing to uppercase letters is an upper function.

```
s = 'Hello, World!'
s.lower()
```

```
'hello, world!'
```

```
s.upper()
```

```
'HELLO, WORLD!'
```

- Capitalize is a function that converts only the first letter into capital letters.

```
s = 'hello'
s.capitalize()
```

```
'Hello'
```

1. Let's Do Coding that Processes Text Data

1.10. Startswith method

- Startswith tells us whether a string starts with a specific character.
- In the code below, the first letter is "A", so it returns True.

```
s = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
s.startswith('A')
```

```
True
```

- By adding the second factor, we can decide where to start finding.

```
s.startswith('B',1)
```

```
True
```

```
s.startswith('B',5)
```

```
False
```

- Since 'B' is in the second position, startswith('B',1) returns True and startswith('B',5) returns False.

1. Let's Do Coding that Processes Text Data

1.11. endswith method

- endswith tells us whether a string ends with a particular character.
  - In the code below, since the last character ends with 'Z', endswith('Z') returns True.

```
s = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
s.endswith('Z')
```

```
True
```

1. Let's Do Coding that Processes Text Data

1.11. endswith method

- endswith tells us whether a string ends with a particular character.
- As a second factor of the endswith method, you can specify the start of a string and end of the string as a third factor.
  - endswith(Search string, start of string, end of string)

```
1 s.endswith('C', 1, 3)
```

```
True
```

```
1 s.endswith('C', 1, 10)
```

```
False
```

Line 1

- s = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' , if 1 and 3 specify the beginning and end of the string, it becomes 'ABC'.
- Therefore, endswith('C', 1, 3) is True.

1. Let's Do Coding that Processes Text Data

1.12. Endswith method

- Replace can replace a specific character in a string with another string.
  - The method of using the replace is as following.
  - replace(old, new, [count]) -> replace("Value to find", "Value to change", [number of change])
  - The code below is a code that removes or replaces commas (,) in the text string as many times as possible.

```
text = '123,456,789,999'

replaceAll= text.replace(",","")
replace_t1 = text.replace(",", "",1)
replace_t2 = text.replace(",", "",2)
replace_t3 = text.replace(",", "",3)
print("Result :")
print(replaceAll)
print(replace_t1)
print(replace_t2)
print(replace_t3)
```

```
Result :
123456789999
123456,789,999
123456789,999
123456789999
```

Unit 14. Dictionary Method1

Pair programming

Pair Programming Practice

Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.

Pair Programming Practice

Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

Collaborative learning environment

A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor's question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students "divide and conquer." When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

Pair programming

Q1. Let's upgrade the program to manage the inventory of convenience stores that we solved in paper coding. In other words, add code to increase or decrease inventory. Also, make simple menus such as inventory inquiry, warehousing, and shipment.

```python
items = {"Coffee": 7, "Pen":3, "Paper cup": 2, "Milk": 1, "Coke": 4, "Book":5}
```

Output example

```
Select menu 1)check stock 2)warehousing 3) release 4) exit :1
[check stock] Enter item: milk
Stock: 1
Select menu 1)check stock 2)warehousing 3) release 4) exit :3
[Release] Enter item and quantity: coke 1
Select menu 1)check stock 2)warehousing 3) release 4) exit :1
[check stock] Enter item: coke
Stock: 3
Select menu 1)check stock 2)warehousing 3) release 4) exit :4
Program exited.
```

Unit 15.

Dictionary Method 2

Unit learning objective (1/3)

Learning objectives

- ✓ Be able to convert list and tuple to dictionary using dic function.
- ✓ Be able to use the dict.defaultdict method to make the key to have a default value even if there is no key.
- ✓ Be able to print dictionary's item using repetition statement.
- ✓ Be able to define dictionaries within dictionaries and assign and copy items inside them to other variables.

Unit learning objective (2/3)

Learning overview

- ✓ Learn how to convert lists and tuples into dictionaries.
- ✓ Learn how to print dictionaries using repetition statements.
- ✓ After creating a dictionary (double dictionary) within the dictionary, learn how to assign or copy items in this overlapping dictionary to other variables.

Concepts you will need to know from previous units

- ✓ How to allocate values by accessing the dictionary key and do basic operations of the dictionary
- ✓ How to add or save values using dictionary's setdefault and get
- ✓ Understanding the reference of the list and the copy of the object

Unit learning objective (3/3)

Keywords

- defaultdict
- Print Dictionary
- Double Dictionary
- Copy Dictionary

Unit 15. Dictionary Method 2

Mission

1. Real world problem

1.1. The development of information and communication technology and telephone dictionary

- Due to the development of information and communication technology, mobile phones are being used in various applications.
- With the spread of COVID-19 and the increase in the number of vaccinated people, the application of mobile phones is gradually expanding, with the introduction of COVID-19 passports as shown in the figure.
- However, the most basic function of a mobile phone is the voice call function, and your phone will contain the names and phone numbers of acquaintances.
- Now we're going to do a mission to make a simple telephone dictionary .

1. Real world problem

1.2. What is phone book?

- The telephone dictionary can be said to be a book in which the phone number and name of the telephone subscriber, or company name and address are posted together.
- Twenty years ago, voice phone functions were the most common means of communication. For this reason, phone booklets have become the most common and universal advertising tool.
- Currently, the effectiveness of telephone dictionary has decreased a lot due to the development of the Internet, but Internet portal services are playing the previous role of telephone dictionary instead.

1. Real world problem

1.3. Additional data on the telephone dictionary

- To learn more about telephone directory, visit https://en.wikipedia.org/wiki/Telephone_directory.

2. Solution

2.1. Telephone directory

- In this mission, we will perform the task of entering review information for restaurants that appear when searching the Internet.
- To simplify the problem, assume that there are three restaurants on the list of telephone directory you will create.
- There are names and phone numbers of each restaurant.
- Write a program that adds an address and review to this restaurant.
- (Constraint: We have not yet learned how to handle complex strings, so when writing addresses and reviews, we do not space them on internal strings.)

3. Mission

3.1. How telephone directory works

```python
tele_dir_dic = {'Chinese restaurant': ('010-1234-4500'), 'Japanese restaurant': ('010-2230-6540'), 'Italian restaurant':('010-3232-7788
print("previous number")
for key, value in tele_dir_dic.items():
    print(key + " :", value)
for key, value in tele_dir_dic.items():
    adr, review = input(key+ 'Please write address and review of: ').split()
    update = (value, adr, review)
    tele_dir_dic[key] = update
print("Updated number")
for key, value in tele_dir_dic.items():
    print(key + " :", value)
```

```
previous number
Chinese restaurant : 010-1234-4500
Japanese restaurant : 010-2230-6540
Italian restaurant : 010-3232-7788
Chinese restaurantPlease write address and review of: Northern_Busan It_was_ok
Japanese restaurantPlease write address and review of: Sasang_Busan It_was_delicious
Italian restaurantPlease write address and review of: Donglae_Busan Very_delicious
Updated number
Chinese restaurant : ('010-1234-4500', 'Northern_Busan', 'It_was_ok')
Japanese restaurant : ('010-2230-6540', 'Sasang_Busan', 'It_was_delicious')
Italian restaurant : ('010-3232-7788', 'Donglae_Busan', 'Very_delicious')
```

3. Mission

3.2. Programming Plan

Pseudocode

[1] Start

[2] The name and phone number of the restaurant are stored in the telephone dictionary.

[3] for I in list in the telephone dictionary do{

[4] Input address and review.
Add address and review to the corresponding restaurant.

}

[5] Print the updated telephone directory.

[6] End

Flowchart

3. Mission

3.3. Telephone directory final code

```
tele_dir_dic = {'Chinese restaurant': ('010-1234-4500'), 'Japanese restaurant': ('010-2230-6540'), 'Italian restaurant':('010-3232-7788
print("previous number")
for key, value in tele_dir_dic.items():
    print(key + " :", value)
for key, value in tele_dir_dic.items():
    adr, review = input(key+ 'Please write address and review of: ').split()
    update = (value, adr, review)
    tele_dir_dic[key] = update
print("Updated number")
for key, value in tele_dir_dic.items():
    print(key + " :", value)
```

Unit 15. Dictionary Method 2

Key concept

1. Converting List and Tuple to Dictionary

1.1. Several ways to make dictionary

- Let's make a dictionary from a list and tuple that already exist.
- You can create a dictionary with elements of a list and tuple as keys using the fromkeys method in the dictionary class dict.
- In this case, all values are initialized to None, and there is also a method of specifying this initial value.

1. Converting List and Tuple to Dictionary

1.2. Make dictionary using fromkeys method

- Find out how to use the dict.fromkeys method using the example below.

List Example

```
keys = ['a', 'b', 'c', 'd']
x = dict.fromkeys(keys)
x
```

```
{'a': None, 'b': None, 'c': None, 'd': None}
```

- First, prepare a list containing keys such as keys=['a', 'b', 'c', 'd'].
- When you put a list containing keys in the dict.fromkeys, a dictionary is created.
- dict.fromkeys creates a dictionary as a key list, and all values are initialized to None.

Tuple Example

```
keys = ('a', 'b', 'c', 'd')
x = dict.fromkeys(keys)
x
```

```
{'a': None, 'b': None, 'c': None, 'd': None}
```

- It is also possible to make dictionaries from tuples.

1. Converting List and Tuple to Dictionary

1.2. Make dictionary using fromkeys method

Focus If a key list and value are added as factors like dict.fromkeys(key list, value), the value is stored as the default value of the key. Let's compare the differences with the diagram below.

```
y = dict.fromkeys(keys, 100)
y
```

```
{'a': 100, 'b': 100, 'c': 100, 'd': 100}
```

Result of dict.fromkeys(keys)
key and default value of the dictionary

Result of dict.fromkeys(keys, 100)
key and default value of the dictionary

1. Converting List and Tuple to Dictionary

1.3. Why do we need defaultdict method?

Let's look at cases where KeyError occurs in dictionary.

An error occurs when you try to reach a key that does not exist inside the dictionary(dic).

```
x = {'a': 0, 'b': 0, 'c': 0, 'd': 0}
x['z']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[9], line 2
      1 x = {'a': 0, 'b': 0, 'c': 0, 'd': 0}
----> 2 x['z']

KeyError: 'z'
```

- What should I do to prevent such an error from occurring? In this case, the defaultdic method is used.
- defaultdict does not print error even if you try to reach a key that does not exist and returns a default value.
- defaultdict is included in the collections module.

1. Converting List and Tuple to Dictionary

1.4. How to use defaultdict

- Following creates a default dictionary with a default value of zero.

```
from collections import defaultdict # bring defaultdic from collections module
keys = ('a', 'b', 'c', 'd')
y = dict.fromkeys(keys, 100)
y = defaultdict(int)    # set default value as int
```

```
y['z']
```

```
0
```

- There is no key 'z' in dictionary y, but if you look up the value of the key, such as y['z'], you will find zero.
- This is because the default value of the int() function is 0.

```
y
```

```
defaultdict(int, {'z': 0})
```

```
int()
```

```
0
```

1. Converting List and Tuple to Dictionary

1.4. How to use defaultdict

- Following creates a default dictionary with a default value of zero.
  - As shown in the following figure, the key has no item for 'z', so it has a default value of 0.

```python
from collections import defaultdict
keys = ('a', 'b', 'c', 'd')
y = dict.fromkeys (keys, 100)
y = defaultdict(int, y) # int as set default value
y['z']
```

```
0
```

Result of y = defaultdict(int)
key and default value of dictionary

1. Converting List and Tuple to Dictionary

1.4. How to use defaultdict

- In Unit 14, we looked at the setdefault method required to process when there is no value for the key.
- This time, let's look at the case of processing using the defaultdict class of the Python built-in

```
from collections import defaultdict

word = 'abcbaabca'
counter = defaultdict(int)
for letter in word:
    counter[letter] += 1

counter
```

```
defaultdict(int, {'a': 4, 'b': 3, 'c': 2})
```

- This code shows the frequency of appearance of each alphabet in the string word as a key-value.
- The defaultdict class automatically calls a function (int) that has passed over to the creator's factor and sets it as the output value if there is no value for all keys.

1. Converting List and Tuple to Dictionary

1.5. Interpretation of defaultdict code

```python
from collections import defaultdict

word = 'abcbaabca'
counter = defaultdict(int)
for letter in word:
    counter[letter] += 1

counter
```

```
defaultdict(int, {'a': 4, 'b': 3, 'c': 2})
```

```python
word = 'abcbaabca'
```

The counter of this code stores the frequency of appearance of each alphabet.

1. Converting List and Tuple to Dictionary

1.6. Precautions of defaultdict

Focus If called without putting anything in int, it returns zero.

- You may wonder why the default value is zero when you put int like defaultdict(int).
- int converts a real number or string into an integer but returns zero if called without putting anything in the int as following.

```
int()
```

```
0
```

- defaultdict may include a function that returns a specific value.
- defaultdict(int) is a function that generates default values to specify int, resulting in zero.
- If you want to set a value other than zero as the default value, you can create and insert a default generation function as following.
- In the following case, the space list becomes a default value.

```
list_dict = defaultdict(list)
list_dict
```

```
defaultdict(list, {})
```

```
list_dict['list1']
```

```
[]
```

2. Print Dictionary Using Loop Statements

2.1. Printing using for iteration statement

- Print the variable i using print after specifying dictionary x in the for iteration statement like for i in x:.
- In this case, the value is not printed, but only the key is.
- What should we do to print out both the key and the value?

```
x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
for i in x:
    print(i, end=' ')
```

```
a b c d
```

- To this end, the dictionary should be designated after for in and the items method should be used as following.

```
for Key, value in dictionary.items():
        Code to repeat
```

2. Print Dictionary Using Loop Statements

2.2. Example of printing using for iteration statement

Ex The following is a code that outputs all keys and values in the dictionary using the for loop.

```
1 x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
2 for key, value in x.items():
3     print(key, value)
```

```
a 10
b 20
c 30
d 40
```

Line 1, 2

- for key, value in x.items: takes the key-value pair from dictionary x and store the key in key and value in value, and repeat the code every time you take it out.
- Therefore, if a key and a value are printed using print, both key-value pairs may be printed.

2. Print Dictionary Using Loop Statements

2.2. Example of printing using for iteration statement

Ex Now print only key of the dictionary as shown below.

```
1 x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
2 for key in x.keys():
3     print(key, end=' ')
```

```
a b c d
```

Line 2

- Key of dictionary x can be called using x.keys method.

2. Print Dictionary Using Loop Statements

2.2. Example of printing using for iteration statement

Ex Print only value of the dictionary.

```
1 x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
2 for value in x.values():
3     print(value, end=' ')
```

```
10 20 30 40
```

Line 2

- Value of the dictionary x can be called using x.values method.

3. Dictionary in Dictionary

3.1. Example code of double dictionary

- A dictionary may be included inside the dictionary in the same manner as dictionary = {key1: {keyA: keyA}, key2: {keyB: keyB}}.
- Write the radius, mass, and orbital cycles of a terrestrial planet in dictionary.

```
terrestrial_planet = {
    'Mercury': {
        'mean_radius': 2439.7,
        'mass': 3.3022E+23,
        'orbital_period': 87.969
    },
    'Venus': {
        'mean_radius': 6051.8,
        'mass': 4.8676E+24,
        'orbital_period': 224.70069,
    },
    'Earth': {
        'mean_radius': 6371.0,
        'mass': 5.97219E+24,
        'orbital_period': 365.25641,
    },
    'Mars': {
        'mean_radius': 3389.5,
        'mass': 6.4185E+23,
        'orbital_period': 686.9600,
    }
}

print(terrestrial_planet['Venus']['mean_radius'])
```

```
6051.8
```

3. Dictionary in Dictionary

3.1. Example code of double dictionary

- The dictionary terrestrial_planet contains keys 'Mercury', 'Venus', 'Earth', and 'Mars'.
- These keys again have dictionary in the value portion.
- That is, the double dictionary is useful when storing hierarchical data.
- To access the dictionary contained in the dictionary, place [ ] (square) after the dictionary by steps as shown below and specify the key.

```
dictionary[key][key]
dictionary[key][key] = value
```

- Here, the dictionary consists of two steps, so square brackets are used twice.
- To output the radius of Venus, find Venus first as following and get the value of mean radius.

```
1 print(terrestrial_planet['Venus']['mean_radius'])
```

```
6051.8
```

4. Assignment and Copying of Double dictionaries

4.1. Assignment and copying

Let's learn about assignment and copying.

- If you create a dictionary and assign it to another variable, such as y = x, it seems that there will be two dictionaries, but actually there is one dictionary.
- Comparing x and y with is operators results in True. That is, only the name of the variable is different, but dictionaries x and y are the same object.

```
x = {'a': 0, 'b': 0, 'c': 0, 'd': 0}
y = x
```

```
x is y
```

```
True
```

x and y refer to the same object.

4. Assignment and Copying of Double dictionaries

4.1. Assignment and copying

Focus Although the names of the variables are different, they point to the same object, so if the value of one variable changes, the value of the other variable also changes.

Since x and y refer to the same object, changing the value of the key 'a' as y['a'] = 99 is reflected in both dictionaries x and y.

```
y['a'] = 99
```

```
y
```

```
{'a': 99, 'b': 0, 'c': 0, 'd': 0}
```

```
x
```

```
{'a': 99, 'b': 0, 'c': 0, 'd': 0}
```

x and y refer to the same object.

4. Assignment and Copying of Double dictionaries

4.3. The result of copying using copy

- Let's use copy so that the objects indicated by the two variables are different objects.
  - Although the names of the variables are different, they point to the same object.
  - As in the list, in order to make two variables completely two, you must copy the key-value with the copy method.

```
x = {'a': 0, 'b': 0, 'c': 0, 'd': 0}
```

```
y = x.copy()
```

- Now comparing x and y with is operators returns False.
- In other words, the two dictionaries are different objects. However, the copied key-value pair is the same, so if you compare them with ==, True is returned.

```
x is y
```

```
False
```

```
x == y
```

```
True
```

As a result of performing y = x.copy, x and y refer to different objects.
But the value of the element is the same.

4. Assignment and Copying of Double dictionaries

4.3. The result of copying using copy

Use copy to make the objects indicated by the two variables be different objects.

```
x = {'a': 0, 'b': 0, 'c': 0, 'd': 0}
```

```
y = x.copy()
y['a'] = 99
```

```
x is y
```

```
False
```

```
x == y
```

```
False
```

- Now comparing x and y with is operator returns False. Let's assign y['a'] = 99.
- That is, since the two dictionaries are different objects and their contents are different, both x is y and x==y return False.

As a result of performing y = x.copy, x and y refer to different objects.
y['a'] = 99, x and y have different element values.

4. Assignment and Copying of Double dictionaries

4.3. The result of copying using copy

- Double dictionaries cannot be copied completely using copy methods.
  - Is it possible to copy the double dictionary containing dictionary in the dictionary with the copy method?
  - Create a double dictionary as following and copy it with the copy method.

```
x = {'a': {'python': '2.7'}, 'b': {'python': '3.6'}}
y = x.copy()
```

  - Now, changing the value of y, such as y['a']['python'] = '2.7.15', is reflected in both x and y.
  - It shows that copying of double dictionaries is impossible with a copy method.

```
y['a']['python'] = '2.7.15'
x
```

```
{'a': {'python': '2.7.15'}, 'b': {'python': '3.6'}}
```

```
y
```

```
{'a': {'python': '2.7.15'}, 'b': {'python': '3.6'}}
```

4. Assignment and Copying of Double dictionaries

4.4. Deepcopy

Focus To completely copy the double dictionary, the deepcopy function must be used.

- To completely copy the double dictionary, the deepcopy function of the copy module should be used instead of the copy method.

```
x = {'a': {'python': '2.7'}, 'b': {'python': '3.6'}}
import copy              # call copy module
y = copy.deepcopy(x)     # deep copy using deepcopy function
y['a']['python'] = '2.7.15'
x
```

```
{'a': {'python': '2.7'}, 'b': {'python': '3.6'}}
```

```
y
```

```
{'a': {'python': '2.7.15'}, 'b': {'python': '3.6'}}
```

- Now, changing the value of dictionary y does not affect dictionary x.
- The copy.deepcopy function provides deep copy of all dictionaries contained in the double dictionary.
- Deep copy is a copy technique that not only shares the address of data, but also creates and stores a copy of data in a separate memory.

Unit 15. Dictionary Method 2

Paper coding

Try to fully understand the basic concept before moving on to the next step.

Lack of understanding basic concepts will increase your burden in learning this course, which may make you fail the course.

It may be difficult now, but for successful completion of this course we suggest you to fully understand the concept and move on to the next step.

Paper coding

Q1. A tuple called study_tup, which has three pairs of elements: student ID number, name, and phone number, exists as shown below. Modify the student_tup below to create and print a dictionary of the pair {student ID number : [name, phone number]}.

Condition for Execution	student_tup = (('211101', 'David Doe', '010-1234-4500'), ('211102', ' John Smith', '010-2230-6540'), ('211103', ' Jane Carter', '010-3232-7788'))
Time	7 min

Output example

```
{'211101' : ['David Doe', '010-1234-4500'] }
{'211102' : ['John Smith', '010-2230-6540'] }
{'211103' : ['Jane Carter', '010-3232-7788'] }
```

Write the entire code and the expected output results in the note.

Paper coding

Q2. Write a bachelor's information program using the student_tup above to receive the student's student ID number as input and print the student's name and phone number.

Example program execution results.	`Enter student ID number : 211101`
	`Name : David Doe`
	`Phone number : 010-1234-4500`

Time	5 min

Write the entire code and the expected output results in the note.

Unit 15. Dictionary Method 2

Let's code

1. Repeat using dictionary methods

1.1. The return form of items

- Items returned through the phone_book.items method are separated by commas in parentheses like ('Name','Hong Gil-dong').
- The data type of the item returned by the items method is a tuple.

```
phone_book = {"David Doe": "010-1234-5678", "John Smith": "010-1234-5679", "Jane Carter": "010-1234-5680"}
```

```
phone_book.items()
```

```
dict_items([('David Doe', '010-1234-5678'), ('John Smith', '010-1234-5679'), ('Jane Carter', '010-1234-5680')])
```

1. Repeat using dictionary methods

1.2. keys method

- Keys is a method of getting all the keys of the dictionary.
  - Items can be found with indexes in the list, but values can be found only with keys in the dictionary.
  - To print all keys used in the dictionary, use the method keys as shown below.

```
phone_book.keys()
```

```
dict_keys(['David Doe', 'John Smith', 'Jane Carter'])
```

1. Repeat using dictionary methods

1.3. values method

- Values is a method that brings all the dictionary values.
  - However, use values to print all values used in the dictionary.

```
phone_book.values()
```

```
dict_values(['010-1234-5678', '010-1234-5679', '010-1234-5680'])
```

  - In addition, items may be used to print all values inside the dictionary.
  - When a function such as phone_book.item is called in the for loop, the (key, value) tuple returns, so it can be used as shown below.

```
phone_book.items()
```

```
dict_items([('David Doe', '010-1234-5678'), ('John Smith', '010-1234-5679'), ('Jane Carter', '010-1234-5680')])
```

```
for name, phone_num in phone_book.items():
    print(name,':', phone_num)
```

```
David Doe : 010-1234-5678
John Smith : 010-1234-5679
Jane Carter : 010-1234-5680
```

2. Module Definition and Import Statement

2.1. What is module?

- Module
  - It is a script file of Python functions, variables, or classes.
  - Python has many modules developed by numerous developers.
  - When importing the created module, write the module name with 'import'.
  - When using it, mark a dot (.) in the module name as shown below and write components in the

```
import [module name].[class name].[method name]
```

- Keyword that brings module is import.

```
import module name 1[, module name2, …]
```

2. Module Definition and Import Statement

2.1. What is module?

Let's look at modules that are needed to learn Python.

- The module provided with the Python installation is called the Python Standard Library.
- There are more than 100 standard libraries, including modules for string and text processing, binary data processing, date, time, and arrangement, numerical operations and mathematical function modules, file and directory access, modules for Unix system database access, data compression and graphics modules.

2. Module Definition and Import Statement

2.2. A datetime module that can receive and manipulate date and time module

- datetime module
  - A module that can provide and manipulate functions related to date and time.
- In the datetime.datetime.now below, the previous datetime is the name of the module, and the latter datetime is the name of the class.
- The now method returns the current time.

```
1 import datetime          # The following sentence is omitted
2 datetime.datetime.now()
```

```
datetime.datetime(2021, 8, 30, 8, 56, 9, 164997)
```

Line 2

- The datatime.now method returns the current year, month, day, hour, minute, second, and microsecond.

2. Module Definition and Import Statement

2.3. today method

- Let's print the value to which the today method is assigned.
  - The today method of the date class returns the current local date.
  - The today object contains the year, month, and day of the current date, so we could print out each attributes.

```
today = datetime.date.today()
print(today)
```

```
2021-08-30
```

```
today
```

```
datetime.date(2021, 8, 30)
```

```
today.year
```

```
2021
```

```
today.month
```

```
8
```

```
today.day
```

```
30
```

2. Module Definition and Import Statement

2.4. dir function

- Let's print the functions and attributes of the class that the module has using a dir function.
  - The dir function returns all class lists and attributes available in the datetime module.
  - MAXYEAR is the maximum year that the datetime object can express and has a value of 9999.
  - MINYEAR has a value of 1.
  - Classes such as date, datetime, datetime_CAPI, time, timedelta, timezone, and tzinfo have the function of conveniently using date, time, time zone, and time zone information.

```
1 print(dir(datetime))
```

```
['MAXYEAR', 'MINYEAR', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__'
spec__', 'date', 'datetime', 'datetime_CAPI', 'sys', 'time', 'timedelta', 'timezone', 'tzinfo']
```

2. Module Definition and Import Statement

2.5. replace method

- The replace method can change the date or time.
  - Used to change the date or time value in datetime.

```
1 start_time = datetime.datetime.now()
2 start_time.replace(month = 12, day = 25)
```

```
datetime.datetime(2021, 12, 25, 9, 0, 35, 190210)
```

Line 1, 2

- The current date object is assigned to start_time.
- Using the replace method, the month and day of the current date is changed to 12 and 25, respectively.

2. Module Definition and Import Statement

2.6. as syntax

- If you use the as syntax, you can use the module name as an alias.
  - It is cumbersome to write "[Module name].[Class name].[Method name]" by dividing it with dot operators everytime.
  - The as syntax can be used to simply shorten the module name data to alias, dt.

```
import [module name] as [module alias]
```

```
import datetime as dt            # use dt instead of datetime from now on
start_time = dt.datetime.now()
start_time.replace(month = 12, day = 25)
```

```
datetime.datetime(2021, 12, 25, 9, 1, 46, 113098)
```

One More Step

import ~ as

- How to give a nickname to a long module name.
- When using classes or methods within a module, connect them with dots, and if the module name is too long, it may be cumbersome to continue writing the name.
- In this case, a new name of the module may be designated using as.
- Short names that are commonly used are m for math, dt for datetime, rd for random, t for turtle.

```
ex1) import datetime as dt

ex2) import random as rd

ex3) import math as m

ex3) import turtle as t
```

2. Module Definition and Import Statement

2.7. from ~ import ~ statement

- The statement from~import is a method of picking out what you need within a module and bringing it.
- It is form of "from [module name] import [class name].[method name].
- In the datetime module, only the class datetime is included.
- If you write a from statement, you can omit the [module name] that you had to write on the import statement.

```
from datetime import datetime
start_time = datetime.now()
start_time.replace(month = 12, day = 25)
```

```
datetime.datetime(2021, 12, 25, 9, 8, 53, 800714)
```

- "from [module name] import *" means to bring all the variables, functions, and classes in the module.
- '*' means everything.
- The code below brings all classes to a module called datetime.
- Among them, today's date is printed using today of date class (method that returns today's date).

```
from datetime import *
today = date.today()
today
```

```
datetime.date(2021, 9, 7)
```

2. Module Definition and Import Statement

2.8. time module

The time module provides time-related information.

- A module that provides time-related information.
- The UTC which is the start time of the Unix system, 0:0:0 on January 1, 1970, is called epoxy.
- The time system used as a standard in Unix operating systems is also called epoxy time or Unix time.
- The code below informs the time in seconds based on the epoxy time (0:0:0 on January 1, 1970).

```
import time
seconds = time.time()
print('Time after epoxy = ' , seconds)
```

```
Time after epoxy = 1630282234.5054421
```

2. Module Definition and Import Statement

2.8. time module

- Try localtime of time module.
  - The local time method returns the received epoxy time value in the form of a date and time.

```
import time
unix_timestamp = time.time()
local_time = time.localtime(unix_timestamp)
local_time
```

```
time.struct_time(tm_year=2021, tm_mon=9, tm_mday=7, tm_hour=23, tm_min=16, tm_sec=42, tm_wday=1, tm_yday=250, tm_isdst=0)
```

  - The strftime method prints a time value obtained by localtime in a date/time format desired by users.

```
import time
unix_timestamp = time.time()
local_time = time.localtime(unix_timestamp)
print(time.strftime('%Y-%m-%d %H:%M:%S', local_time))
```

```
2021-08-30 09:11:42
```

  - %Y is year, %m is month, %d is day, and '%Y-%m-%d' means that it is 'year-month-day' format'.
  - '%H:%M:%S': Each represents time (24 hours), minutes (00 ~ 59), and second (00 ~ 59).

2. Module Definition and Import Statement

2.9. math module

- Math module that can use math-related functions.
- Math module
  - A module with functions related to mathematics.
  - Pi value, natural constant e value, etc. are defined.
  - Includes math-related functions such as sin, cos, tan, log, pow, ceil, floor, trunc, fabs, copysign(x,y), etc.

2. Module Definition and Import Statement

2.10. Print the built-in functions of the math module using dir

```python
import math
print(dir(math))     # returns built-in functions of the math module
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'ata
nh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factoria
l', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt',
'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder',
'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

2. Module Definition and Import Statement

2.11. Learn the functions of math-related modules math

Let's find out the functions using functions in the module.

```
import math as m
m.pow(3, 3)  # 3 to 3 squares
```

```
27.0
```

```
m.fabs(-99)  # The absolute value of -99
```

```
99.0
```

```
m.ceil(2.1)  # 2.1 rounded up
```

```
3
```

```
m.ceil(-2.1)  # -2.1 rounded up
```

```
-2
```

```
m.floor(2.1)  # 2.1 rounded down
```

```
2
```

```
m.log(2.71828)
```

```
0.999999327347282
```

```
m.log(100, 10) # value of 100 with logarithm 10 as the bottom
```

```
2.0
```

2. Module Definition and Import Statement

2.11. Learn the functions of math-related modules math

Mathematical functions in the math module, description, and examples of use

Function	Description	Example
fabs(x)	Returns the absolute value of real number x.	fabs(-2) → 2.0
ceil(x)	Raises x to a near integer and returns it.	ceil(2.1) → 3, ceil(-2.1) → -2
floor(x)	Rounds down x to a near integer and returns it.	floor(2.1) → 2, ceil(-2.1) → -3
exp(x)	Returns e^x.	exp(1) → 2.71828
log(x)	Returns natural logarithm x.	log(2.71828) → 1.0
log(x, base)	Returns the value of log x with base as base.	log(100, 10) → 2.0
sqrt(x)	Returns the square root of x.	sqrt(4.0) → 2.0
sin(x)	Returns the sine value of x. x in radians.	sin(3.14159/2) → 1, sin(3.14159) → 0
asin(x)	Returns asin of x.	asin(1.0) → 1.57, asin(0.5) → 0.523599
cos(x)	Returns the cos value of x. x in radians.	cos(3.14159/2) → 0, cos(3.14159) → -1
acos(x)	Returns acos of x.	acos(1.0) → 0, acos(0.5) → 1.0472
tan(x)	Returns the tan value of x. x in radians.	tan(3.14159/4) → 1, tan(0.0) → 0
degrees(x)	Changes the angle x from radians to degrees.	degrees(1.57) → 90
radians(x)	Changes the normal angle x to radians.	radians(90) → 1.57

2. Module Definition and Import Statement

2.12. Trigonometric functions using radians

- Let's find out why sin (90) is not 1 in the code below.
  - This is because the trigonometric function in Python's math module uses the radian angle as a factor value.
  - In order to use the general angle of 90 degrees in the math module, it is necessary to convert it to a radian notation like math.pi/2.0.

```
import math as m

m.sin(0.0)      # value of sin 0
```

```
0.0
```

```
m.sin(90.0)      # do not use general angle of 90.0
```

```
0.8939966636005579
```

```
m.sin(m.pi/2)     # radian notation
```

```
1.0
```

Unit 15. Dictionary Method 2

Pair programming

Pair Programming Practice

Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.

Pair Programming Practice

Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

Collaborative learning environment

A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor's question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students "divide and conquer." When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

Pair programming

Q1. The student_tuple list with tuples as elements is as shown below. Tuple, which is the element of this tuple consists of a (student ID number, name, phone number). Using this, make a dictionary for (student ID number: name) and print it out. When inquiring by student ID number, make sure that the student ID number, name, and phone number are printed as shown below.

```
student_tuple = [('211101', 'David Doe', '010-123-1111'), ('211102', 'John Smith', '010-123-2222'),
                 ('211103', 'Jane Carter', '010-123-3333')]
```

Example Output

```
211101 : David Doe
211102 : John Smith
211103 : Jane Carter
Enter student ID number : 211103
211103 student is Jane Carter and phone number is 010-123-333.
```

Unit 16.

Set Data Types

Unit learning objective (1/3)

Learning objectives

- ✓ Be able to create set data types that do not allow duplication of data values.
- ✓ Be able to check a specific value in the set through the in operator.
- ✓ Be able to use the set to create an empty set or change the value of other data into a set data type.
- ✓ Be able to find out the relationship between the elements of the two sets after looking at operations that can be applied to the set, such as union and intersection.
- ✓ Be able to test whether the two sets are the same through comparative operations. It is also possible to test whether it is a true subset or a subset.
- ✓ Be able to easily print sets using iteration statements and put specific values using conditional statements in set expressions.

Unit learning objective (2/3)

Learning overview

- ✓ Learn how to create a set of data types without an order that does not allow duplication of data values.
- ✓ Through the in operator, we learn how to check whether a specific value exists in the set.
- ✓ Learn how to conveniently remove duplicate data using set.
- ✓ Learn convenient operations that can be applied to the set, such as union and intersection.
- ✓ Learn how to obtain subsets and true subsets through comparative operators.

Concepts you will need to know from previous units

- ✓ How to print all elements using the length of each array by further using len, range, in, etc. of sequence objects
- ✓ How to use sequence objects and dictionary methods (del, index, pop, etc.)
- ✓ Knowing the differences in each data type, such as lists, tuples, dictionaries, and strings, and being able to use data types suitable for the situation

Unit learning objective (3/3)

Keywords

- set
- Set calculation
- Set comparison
- Conditional statement

Mission

1. Real world problem

1.1. Find the probability of getting above than a certain number of values when throwing a dice

- In this mission, we will assume that several people are playing Monopoly.
- The board game is played through dice and wins if you roll two dice and move on the board as much as the sum of the two numbers that appeared at the time, buy real estate, rent it to other players, and make all other players bankrupted.
- To win a game, let's find the probability of getting more than a certain number of values.
- Using that probability, it would be advantageous in the game to find and purchase places that might take the most depending on the other person's location.

1. Real world problem

1.2. What is Monopoly?

- Monopoly is a board game that started in the United States in 1903.
- In the game, the player rolls the dice, moves on the board, purchases real estate, receives rent from another player who arrives at the property, monopolizes real estate of the same color, builds, develops, and controls buildings. You can also trade with other players to achieve the goal of monopoly.
- If one gets all other players go bankrupt while receiving rent, the person becomes winner. It is currently produced with copyright by Hasbro, the largest toy company in the United States. This game is still very popular these days.

1. Real world problem

1.3. Learn more about Monopoly

- To learn more about Monopoly, visit https://boardgamegeek.com/boardgame/1406/monopoly.

1. Real world problem

1.4. Monopoly Rules

- In a typical Monopoly game, two dice are used, but we will assume three dice are used here.
- Suppose that three dice with six sides are given.
- We are going to throw this dice and print out the total number of cases that can happen.
- Next, the probability of the sum of the eyes to be greater than or equal to a specific value x will be calculated and printed.
- This problem is the case where (1, 2, 1) and (1, 1, 2) and (2, 1, 1) are considered the same. Therefore, it is a problem that cannot be expressed with a list, tuple, or dictionary data structure.
- It is convenient to use the set data type to solve this mission.

2. Mission

2.1. How Monopoly works

```
cases_3times = set()
for i in range(1, 7):
    for j in range(1, 7):
        for k in range(1, 7):
            cases_3times.add((i, j, k))

            total_cases = len(cases_3times)
print('Event that can happen by rolling the dice 3 times are', total_cases, 'cases.')
```

```
Event that can happen by rolling the dice 3 times are 216 cases.
```

2. Mission

2.1. How Monopoly works

```python
for i in range(3, 19): #Find the probability of getting more than 3 to more than 18
    total_cases = len(cases_3times)
    n_cases = 0
    for c in cases_3times:
        if sum(c) >= i:
            n_cases += 1
    prob = n_cases * 100 / total_cases
    print('probability of getting more than {:2d} is {:6.2f}%'.format(i,prob))
```

```
probability of getting more than  3 is 100.00%
probability of getting more than  4 is  99.54%
probability of getting more than  5 is  98.15%
probability of getting more than  6 is  95.37%
probability of getting more than  7 is  90.74%
probability of getting more than  8 is  83.80%
probability of getting more than  9 is  74.07%
probability of getting more than 10 is  62.50%
probability of getting more than 11 is  50.00%
probability of getting more than 12 is  37.50%
probability of getting more than 13 is  25.93%
probability of getting more than 14 is  16.20%
probability of getting more than 15 is   9.26%
probability of getting more than 16 is   4.63%
probability of getting more than 17 is   1.85%
probability of getting more than 18 is   0.46%
```

2. Mission

2.2. Programming Plan

Pseudocode

[1] Start

[2] Declare the variables that all dice contain..

[3] for I to from1 to 7 do{

[4] for I to from1 to 7 do{

[5] for I to from1 to 7 do{

[6] In the case of dice, everything goes into a tuple shape.

[7] number in cases_3times saved in total_cases.

[8] The number of cases is printed out.

}

[9] End

Flowchart

2. Mission

2.2. Programming Plan

Pseudocode

[1] Start

[2] All dice cases are stored in cases_3times and the number of cases is stored in total_cases.

[3] for I to from 3 to 18 do{

[4] Initialize n_cases, which accumulate and add more than a certain number of eyes, to zero.

[5] for j to all number of cases do{

[6] If the sum of the dice is greater than or equal to I, then{

[7] add number of cases. } }

[8] (number of cases accumulated and added above) * 100/total_cases are given to prob to obtain probability.

[9] print probability. }

[10] End

Flowchart

2. Mission

2.3. Monopoly final code #1

```
cases_3times = set()
for i in range(1, 7):
    for j in range(1, 7):
        for k in range(1, 7):
            cases_3times.add((i, j, k))

            total_cases = len(cases_3times)
print('Event that can happen by rolling the dice 3 times are', total_cases, 'cases.')
```

2. Mission

2.3. Monopoly final code #2

```
for i in range(3, 19): #Find the probability of getting more than 3 to more than 18
    total_cases = len(cases_3times)
    n_cases = 0
    for c in cases_3times:
        if sum(c) >= i:
            n_cases += 1
    prob = n_cases * 100 / total_cases
    print('probability of getting more than {:2d} is {:6.2f}%'.format(i,prob))
```

Key concept

1. Creating Set

1.1. The definition of the set and the method of declaration

- The set data type is a basic data type provided by Python but is not provided as a basic type in C or Java languages.
  - The set referred to in mathematics is a collection of objects that satisfy a clear criterion or a given property.
  - Python's set is an unordered data type, and similar to dictionary, uses {} parentheses.
  - Duplicating items with the same value are not allowed.
  - It is possible to perform a set operation such as an intersection, union, and a difference of set.

- Different ways to declare the set.

Making empty set	set0 = set()
Making basic set	set1 = {1, 2, 3, 4}
Making set from tuple	n_tuple = (1, 2, 3, 4) set2 = set(n_tuple)
Making set from list	n_list = [1, 2, 3, 4] set3 = set(n_list)

1. Creating Set

1.2. How to make sets

- When creating a set, you can also use the set function for a list or tuple that has already been created as shown below.
- Since a set is a collection of elements that are not in an order, the index cannot be used. Therefore, slicing is impossible.

```
days_list = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'] # list
days_set = set(days_list)    # making sets from list
days_set
```

```
{'Fri', 'Mon', 'Sat', 'Sun', 'Thu', 'Tue', 'Wed'}
```

```
fruits_tuple = ('apple', 'orange', 'water melon') #tuple
fruits_set = set(fruits_tuple)    # making sets from tuple
fruits_set
```

```
{'apple', 'orange', 'water melon'}
```

One More Step

Set of string 'hello' can be made using set() function. But the final elements of the set are {'h', 'l', 'e', and 'o'} because the set does not allow duplication of the letter 'l' from the string 'hello'. The result is the same as {'e', 'h', 'l', and 'o}}.

```
h_str = 'hello'          # String
h_set = set(h_str)       # Makina sets from string
h_set                    # Set does not allow duplication of string 'l'
```

```
{'e', 'h', 'l', 'o'}
```

2. Check the specific value in the set

2.1. in operator

To check which items are in the set, you can use the in operator, just like with list.

```
numbers = {2, 1, 3}
if 1 in numbers:        # check if 1 is in the numbers set
    print("1 is in the set.")
```

```
1 is in the set.
```

One More Step

- The in operator is not only used for sets.
  - The in operator used in this section is not an operator that can only be used for sets but can be applied to various data with multiple items. As a representative example, it is possible to check whether there are specific items in the list as shown below.

```
a_list = ['hello', 'world', 'welcome', 'to', 'python']
'python' in a_list # check if string 'python' is in the a list
```

```
True
```

2. Check the specific value in the set

2.2. Indexing in sets

Focus Since there is no index in the elements of the set, indexing or slicing operations are meaningless.

We can add an element using add method.

```
numbers = {1, 2, 3}
numbers.add(4)
numbers
```

```
{1, 2, 3, 4}
```

We can also use the remove method to delete elements in the set.

```
numbers.remove (4)
numbers
```

```
{1, 2, 3}
```

3. Using Set

3.1. Creating sets

- It is also possible to create a set from the list as following.
- However, in the set data type, when elements overlap, they are automatically removed.

```
set([1, 2, 3, 1, 2]) # Create set from list
```

```
{1, 2, 3}
```

- And it is possible to create a set from a string.
- In this case, each character becomes an element.

```
set ("abcdefa")    #Since sting is also sequence type it is possible to convert to sets
```

```
{'a', 'b', 'c', 'd', 'e', 'f'}
```

3. Using Set

3.2. How to declare an empty set

To create an empty set, use the set function as shown below.

```
numbers = set()    # Create an empty set
numbers
```

```
set()
```

4. Set Calculation

4.1. Set operator and method

- Use & for intersection, | for union, - for a difference of set, ^ for symmetric difference of set.
- If there are sets s1 and s2, in order to apply the operations of these sets, we need to express the inclusion relationship with the elements of the sets in a diagram.

4. Set Calculation

4.2. Union calculation

- Use | operator or union method for union calculation.

Calculation	Calculation result diagram
Union	
s1 | s2 = {1, 2, 3, 4, 5, 6, 7, 8, 9} or s1.union(s2)	

4. Set Calculation

4.3. Intersection calculation

Use & operator or intersection method for intersection calculation.

Calculation	Calculation result diagram
Intersection	
s1 & s2 = {4, 5, 6} or s1.intersection(s2)	

4. Set Calculation

4.4. Difference of sets calculation

- Use – operator or difference method for difference of sets.

Calculation	Calculation result diagram
Difference of sets	
s1 - s2 = {1, 2, 3} or s1.difference(s2)	

4. Set Calculation

4.5. Symmetric difference of sets calculation

- Symmetric difference of sets is subtraction of intersection from union, (s1 ∪ s2) - (s1 ∩ s2). Use ^ operator or symmetric_difference method to calculate.

Calculation	Calculation result diagram
Symmetric difference of sets	
s1 ^ s2 = {1, 2, 3, 7, 8, 9} or s1.symmetric_difference(s2)	

4. Set Calculation

4.5. Symmetric difference of sets calculation

```
s1 = {1, 2, 3, 4, 5, 6}
```

```
s2 = {4, 5, 6, 7, 8, 9}
```

```
s1 | s2    # Find union
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
s1 & s2    # Find intersection
```

```
{4, 5, 6}
```

```
s1 - s2    # Find difference of set
```

```
{1, 2, 3}
```

```
s1 ^ s2    # Find symmetric difference of set
```

```
{1, 2, 3, 7, 8, 9}
```

4. Set Calculation

4.6. Several methods of set

Focus Set operators |, &, -, ^ will give same results as union, intersection, difference, symmetric_difference like following.

```
s1 = {1, 2, 3, 4, 5, 6}
```

```
s2 = {4, 5, 6, 7, 8, 9}
```

```
s1.union(s2)        # Find union
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
s1.intersection(s2) # Find intersection
```

```
{4, 5, 6}
```

```
s1.difference(s2)   # Find difference of set
```

```
{1, 2, 3}
```

```
s1.symmetric_difference(s2) # Find symmetric difference of set
```

```
{1, 2, 3, 7, 8, 9}
```

4. Set Calculation

4.7. Continuous calculations are also possible

```
s1 = {1, 2, 3, 4, 5, 6}
```

```
s2 = {4, 5, 6, 7, 8, 9}
```

```
s3 = {5, 6, 9, 10, 11}
```

```
s1 & s2 & s3    # Find union of s1, s2, s3
```

```
{5, 6}
```

```
s1 - s2- s3    # subtract elements of s2 and s3 from s1
```

```
{1, 2, 3}
```

s1 & s2 & s3 = { 5, 6 }

4. Set Calculation

4.8. The methods related to the set and what they do

Method	Function
add(x)	Add element x to the set.
discard(x)	Delete element x from the set.
clear	Delete all elements in the set.
union(s)	Find union with set s. Same as | operator.
difference(s)	Find difference of set with set s. Same as – operator.
intersection(s)	Find intersection of set with set s. Same as & operator.
symmetric_difference(s)	Find symmetric difference of set with set s. Same as ^ operator.
issubset(s)	Find id set s is a subset. Return True/False.
issuperset(s)	Find id set s is a superset. Return True/False.
isdisjoint(s)	Find id set s is a coprime. Return True/False.

4. Set Calculation

4.9. Calculating subset and superset

- Subset: If all elements of a set A belong to another set B, then A is a subset of B.
  ex: A = {1, 2}, B = {1, 2, 3} A is a subset of B.
- Here, B is a superset of A because B contains everything A has.

```
s1 = {1, 2, 3, 4, 5}
s2 = {1, 2, 3}      # is s1's subset
s3 = {1, 2, 6}      # not s1's subset
```

```
s2.issubset(s1)     # method asking if s2 is subset of s1
```

```
True
```

```
s3.issubset(s1)     # method asking if s3 is subset of s1
```

```
False
```

```
s1.issuperset(s2)  # method asking if s1 is superset of s2
```

```
True
```

```
s1.issuperset(s3)   # method asking if s1 is superset of s3
```

```
False
```

s2 is a subset of s1
s3 is not a subset of s1

4. Set Calculation

4.10. Example of calculation of coprime

Ex Coprime calculation of two sets

```
1 s1 = {1, 2, 3}
2 s2 = {10, 20, 30}
3 s1.isdisjoint(s2)    # checking if s1 and s2 are coprime
```

```
True
```

Line 3

- Returns True since two sets are coprime.

One More Step

Product set or Cartesian product

- There are concepts called product set or Cartesian product.
- Product set can be written in Cartesian product $A \times B$
- It become a set of tuples (a,b) that can be made using all the elements a in the set A and all elements b in the set B, just like $\{(a,b) | a \in A,\ b \in B\}$

A		B		AxB
1	X	2	=	(1, 2)
3		4		(1, 4)
				(3, 2)
				(3, 4)

	2	4
1	(1, 2)	(1, 4)
3	(3, 2)	(3, 4)

Set B

Set A

One More Step

Product set of product sets A, B

```
1  A = {1, 3}    # element of set A
2  B = {2, 4}    # element of set B
3  res = set()
4
5  for i in A:
6      for j in B:
7          res = res | {(i, j)}    # product set with double for loop
8  AxB = res    # product set AxB of A and B(not A *B)
9  print('A =', A)
10 print('B =', B)
11 print('AxB =', AxB)    # print product set of A and B
```

```
A = {1, 3}
B = {2, 4}
AxB = {(3, 2), (1, 2), (3, 4), (1, 4)}
```

Line 4-7

- The code above performed a union of elements of (i, j) with res using a double for loop.
- Since the set does not allow duplication, the duplicated elements is removed from the union.
- Tuples containing each element as a component are then completed, and they are not duplicated.
- In the variable AxB, x is not a multiplication operator *.

5. Set Comparison

5.1. Set comparison calculation

- You can also check whether the two sets are the same.
- Using operators == and =! is the easiest way.

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

```
True
```

- Using the < operator and the <= operator, it is possible to check whether the set is a true subset or a subset.
- A true subset refers to a subset of the entire subset excluding one's own.
- As shown in the code below, B is a subset of A, and B is not equal to A. At this time, B is called the true subset.

```
A = {1, 2, 3, 4, 5}
B = {1, 2, 3}
B < A
```

```
True
```

6. Iteration Statement and Sets

6.1. Approach using iteration statement

- Since the set is a repeatable object, a iteration statement can be used.
- Each item in the set can be accessed and printed using the for iteration statement as following.

```
numbers = {2, 1, 3}
for x in numbers:
    print(x, end=" ")
```

```
1 2 3
```

6. Iteration Statement and Sets

6.2. Order of sets

Focus The set may be different from the input order because it has no order.

If you want to print items in sorted order, you can use the sorted function as shown below.

```
for x in sorted(numbers):
    print(x, end=" ")
```

```
1 2 3
```

7. Using Conditional statement in Set Statement

7.1. Use the for statement in the Shorthand

- A set can be created using for loop statements and shorthand as follows.

```
a = {i for i in 'apple'}
```

```
a
```

```
{'a', 'e', 'l', 'p'}
```

- Compare the results with the following list shorthand.

```
Result of above is [ 'a', 'p', 'p', 'l', 'e' ]\.
set([ 'a', 'p', 'p', 'l', 'e'])is
{'a', 'e', 'l', 'p'}.
```

7. Using Conditional statement in Set Statement

7.2. Example of shorthand of the set

- Using shorthand, you can get the desired result with concise coding.
- As shown in the statement below, the if conditional statement in the shorthand is specified after the for loop statement.
- {Equation for variable in set if conditional expression}

```
1 a = {i for i in 'pineapple' if i not in 'apl'}
2 a
```

```
{'e', 'i', 'n'}
```

Line 1

- Using for, each alphabet value of 'pineapple' is assigned to i, which is filtered by the condition of the if statement.
- In the above code, if i does not correspond to the alphabets 'a', 'p', and 'l', then 'e', 'I', and 'n' are stored in a.
- This code is a method of extracting strings excluding the alphabets 'a', 'p', and 'l' from the 'pineapple' string.

Unit 16. Set Data Types

Paper coding

Try to fully understand the basic concept before moving on to the next step.

Lack of understanding basic concepts will increase your burden in learning this course, which may make you fail the course.

It may be difficult now, but for successful completion of this course we suggest you to fully understand the concept and move on to the next step.

Paper coding

Q1. Use the set function to generate and print the set s1 from the next list lst.

Program variable conditions	`lst = ['apple', 'mango', 'banana']  # A list of 3 fruit information` `s1 = {'apple', 'mango', 'banana'}  # Set s1 generated from lst`
Time	7min

Output Example

```
s1 = {'banana', 'apple', 'mango'}
```

Write the entire code and the expected output results in the note.

Paper coding

Q2. Write down the computational results for the following two sets. Find the results from 1) to 7).

Program operation conditions	`s1 = {10, 20, 30, 40}`
	`s2 = {30, 40, 50, 60, 70}`
	`1) s1 ¦ s2`
	`2) s1 & s2`
	`3) s1 – s2`
	`4) s1 ^ s2`
	`5) s1.issubset(s2)`
	`6) s1.issuperset(s2)`
	`7) s1.isdisjoint(s2)`

Time	5min

Write the entire code and the expected output results in the note.

Unit 16. Set Data Types

Let's code

1. If the Order is Not Important : Set

1.1. Definition of set

- The set used in Python is also similar to the set of mathematics, and unlike tuples, it is a data type without order.
- In particular, duplication of items with the same value is not allowed, and various set operations such as intersection, union, difference of set, and symmetric difference of set can be performed.

```
numbers = {2, 1, 3} # set data type consisted of 3 numbers
numbers
```

```
{1, 2, 3}
```

1. If the Order is Not Important : Set

1.2. Various functions that can be applied to a set

- Many functions can be used to process information about items in a set.
- Functions such as len, max, min, stored, sum, etc. may also be used for the set.
- Let's make a set of six items as shown below. Apply various functions.

```
a_set = {1, 5, 4, 3, 7, 4 }     # make set of six items
len(a_set)                      # number of item is 5 excluding duplication
```

```
5
```

```
max(a_set)                      # largest number within the item is 7
```

```
7
```

```
min(a_set)                      # smallest number within the item is 1
```

```
1
```

```
sorted(a_set)                   # make list by arranging the items, without duplication
```

```
[1, 3, 4, 5, 7]
```

```
sum(a_set)                      # since duplicated number is only used once, sum is 20
```

```
20
```

1. If the Order is Not Important : Set

1.3. Logical computational functions applicable to a set

- Logical operations that can be applied to data with multiple items include all and any functions.
- This function comprehensively returns the result of boolean evaluation for each item of data given as a factor.
- Let's make a set of six items as shown below. Apply various functions.

```
a_set = { 1, 0, 2, 3, 3}
all(a_set), any(a_set)    # Is a_set all true? Is there - in a_set? check
```

```
(False, True)
```

- all evaluates each of the repeatable data items inside in a boolean type and returns true if everything is true.
  - If one is false, it returns false.
- any evaluates each of the repeatable data items in the form of a boolean and returns true regardless of the other value if any is true.
  - If everything is false, returns false.

1. If the Order is Not Important : Set

1.4. Aliquot and greatest common factor and programming thinking

- Let's think about how to get the aliquot of two integers.
- The aliquot of 10 can be 1, 2, 5, and 10, of which 1 and 2 and 5 excluding their own 10 are called true aliquot.
- Make a list as following, and divide ten with two, three, four, five...Divide all the numbers up to 9 and check if the remaining values are zero, and if so, put them in the list.

```
1 num = 10
2 divisors = []
3
4 for i in range(2, num):
5     if num % i == 0:
6         divisors.append(i)
7
8 print(num,'s true aliquot :', divisors)
```

```
10 s true aliquot : [2, 5]
```

Line 4, 5, 6

- i increases by 1 from 2 and loops up to 9. The reason for starting with 2 is to get the true aliquot.
- Divide num by i and add it to the list because it is a true divisor if the remainder is zero.
- You can see the aliquot numbers 2 and 5 of 10.

1. If the Order is Not Important : Set

1.4. Aliquot and greatest common factor and programming thinking

- To use this code to find the maximum common divisor of 48 and 60, find the true divisor of 48 and 60, and then find the largest of them.
- To this end, let's declare an empty set of data types of get_divisors1 and divisors2 and add a value to this set where num % i becomes zero.

```
num1 = 48
divisors1 = set()

for i in range (2, num1):
    if num1 % i == 0:
        divisors1.add(i)
print(num1, 's true aliquot : ', divisors1)

num2 = 60
divisors2 = set()

for i in range (2, num2):
    if num2 % i == 0:
        divisors2.add(i)
print (num2, 's true aliquot :', divisors2)
```

```
48 s true aliquot :  {2, 3, 4, 6, 8, 12, 16, 24}
60 s true aliquot : {2, 3, 4, 5, 6, 10, 12, 15, 20, 30}
```

1. If the Order is Not Important : Set

1.4. Aliquot and greatest common factor and programming thinking

- Now let's find a common divisor among these divisors.
- And use the max function to find the largest of these values.
- Of course, there is Euclid's method of obtaining two maximum divisors faster than this method, but it would be a programmatic way of thinking to find solutions through a step-by-step process of solving problems like this.

```
1 print(divisors1.intersection (divisors2))
2 print(num1, num2, '\'s maximum divisor : ', max(divisors1.intersection(divisors2)))
```

```
{2, 3, 4, 6, 12}
48 60 's maximum divisor :  12
```

Line 2

- Find the intersection of the elements A and B and then print the largest value using the max function.
- The maximum common divisor 12 of 40 and 60 is printed.

2. Aggregation Using Zip Function

2.1. The characteristics of the zip function and repeatable data types

- Data types such as lists, dictionaries, sets, and tuples are called repeatable data types.
- When several repeatable data types are entered, the function that returns the tuple iterator by combining them is the zip function.
- The zip function can receive several repeatable data types as shown below. This is called aggregation.

```
zip(*iterables)
```

2. Aggregation Using Zip Function

2.2. Example code using zip function

```
empty_iterator = zip()
result = set(empty_iterator)
result
```

```
set()
```

- If the factor is not transferred, the zip function returns a space iterator.
- When one repeatable data type factor enters, the zip function returns a repeater for one tuple.
- Each element of this tuple is transferred one value obtained from the data transferred to the factor.
- When n repeatable data-type factors enter, the tuple iterator becomes a tuple iterator with as many tuples as the number of factors.
- Here, the i-th element, the tuple, is composed of i-th elements of each repeatable data type factor that has passed through to the factor.

2. Aggregation Using Zip Function

2.3. Aggregation example using zip function

```python
a = [10, 20, 30]                # list a
b = ('ten', 'twenty', 'thirty') # tuple b
for val in zip(a, b):           # print tuple created by aggregating list a and tuple b
    print(val)
```

```
(10, 'ten')
(20, 'twenty')
(30, 'thirty')
```

2. Aggregation Using Zip Function

2.4. Precautions for using zip function

Focus When one repeatable data type factor is transferred to the zip function, it appears as a tuple object ('a',), not 'a'.

- If the list with a string as an element is entered to the zip function as repeatable data.
- Note that the resulting element is a tuple, so it appears as a tuple ('a',) with one element called 'a', not just 'a' because it is a tuple.

```
lst = ['a', 'b', 'hello', 'this']
my_iterator = zip(lst)
result = set(my_iterator)
result
```

```
{('a',), ('b',), ('hello',), ('this',)}
```

2. Aggregation Using Zip Function

2.5. Zip aggregation example

- Aggregate in case of delivering multiple repeatable data-type factors. What things are thrown away here?

```
str_list = [ 'hello', 'world', 'python', 'rocks']
int_tuple = (1, 2, 3)
int_list = [400, 500, 600, 700, 800]
my_iterator = zip(str_list, int_tuple, int_list)
list(my_iterator)
```

```
[('hello', 1, 400), ('world', 2, 500), ('python', 3, 600)]
```

2. Aggregation Using Zip Function

2.6. Zip and tuple unpacking

- Unpacking is the operation of returning tuples tied through a zip function to their original repeatable data.
- Instead of writing any other function, you need to enter the data bound to the zip function as a factor and put the * operator in front of the factor.

```
zip_lst = list(zip(str_list, int_tuple, int_list))
a, b, c = zip(*zip_lst)     # unpack tuple
print(a, b, c)
```

```
('hello', 'world', 'python') (1, 2, 3) (400, 500, 600)
```

Pair programming

Pair Programming Practice

Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.

Pair Programming Practice

Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

Collaborative learning environment

A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor's question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students "divide and conquer." When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

Pair programming

Q1. There is a list mylist with tuple(m, n) as elements as shown below. If there is tuple with (a,b) value and a, b values entered from the user, print 'There is an element (a, b) in xth'. If there is no (a, b) but (b, a) is there, print 'There is no (a, b) but (b, a) there is at yth. If there is no (a, b) nor (b, a), print 'There is no such element'.

mylist = [(1, 2), (4, 5), (4, 2), (3, 1), (9, 4)]

Output Example

```
Enter two integers: 1 2
There is (1,2) at the first.
```

```
Enter two integers: 5 4
There is no (5,4) but there is (4,5) at the second.
```

```
Enter two integers: 3 9
There is no (3,9) nor (9,3)
```

Cofinanciado por la Unión Europea
GOBIERNO DE ESPAÑA MINISTERIO DE INDUSTRIA Y TURISMO
EOI Escuela de Organización Industrial
Fondos Europeos
Sistema Nacional de Garantía Juvenil
SAMSUNG

End of Document

Samsung Innovation Campus

Cofinanciado por la Unión Europea
GOBIERNO DE ESPAÑA MINISTERIO DE INDUSTRIA Y TURISMO
EOI Escuela de Organización Industrial
Fondos Europeos
Sistema Nacional de Garantía Juvenil
SAMSUNG

SAMSUNG

Together for Tomorrow!

Enabling People

Education for Future Generations

ⓒ2023 SAMSUNG. All rights reserved.

Samsung Electronics Corporate Citizenship Office holds the copyright of book.

This book is a literary property protected by copyright law so reprint and reproduction without permission are prohibited.
To use this book other than the curriculum of Samsung Innovation Campus or to use the entire or part of this book, you must receive written consent from copyright holder.