

Laboratory Activity No. 4

Sequence and Mapping Types

Course Code: CPE103

Program: BSCPE

Course Title: Object-Oriented Programming

Date Performed: 08-02-2025

Section: 1A

Date Submitted: 15-02-2025

Name: Ruperto, April Anne A.

Instructor: Maria Rizette H. Sayo

1. Objective(s):

This activity aims to familiarize students in implementing Sequence and Mapping Types in Python.

2. Intended Learning Outcomes (ILOs):

The students should be able to:

2.1 Create a Python program that can change its output based on different conditions

2.2 Use the different iterative statements in a Python program

3. Discussion:

Python has data types that are called Sequence Types and Mapping Types. **Sequence types** are data types composed of items or elements that can be accessed through index values or iterative statements. The sequence types are: **lists**, **tuples**, **ranges**, and **texts** or **strings**. For Mapping Types, there is only currently one standard mapping type which is the **dictionary**. The dictionary data type is created using **key:value** pairs and multiple key:value pairs can be created under one dictionary using commas. These data types will be explored further in the activity.

Lists

Python lists are the equivalent of arrays and arraylists in other programming language. The size of Python lists can increase or decrease dynamically meaning items or elements can continuously be added or removed from a list. A Python list can contain elements or items of different types unlike in compiled languages. A Python list can contain values of any data type in Python and can be accessed either through the index of the elements or a loop. The value of the index can be modified which is also referred to as Mutable Property.

```
[1,2.0,-3,'Hello',True,['another','element'],(1,2,3)]
```

Strings

Strings are composed of individual characters concatenated together to form strings. Each character is considered an element of the string and has an index number that maps to the specific value like in lists. Strings can be accessed through either index number(s) or a loop. Unlike the list however, the indexes of a string cannot have its value modified and deleted which is referred to as Immutable Property.

```
"Hello World" 'Hello World' 'a' 'b' 'ab' """Hello World""" 'Hello\n World'
```

Tuples

Tuples are similar to Python lists in the way they are accessed through indexes and loops. However, unlike lists, tuples cannot have its values modified and deleted which is referred to as Immutable Property. Tuples can only be concatenated with other Tuples.

```
(1,2.0,-3,'Hello',True,['another','element'],(1,2,3))
```

Ranges

The range type represents an immutable sequence of numbers and is commonly used as an incrementor in for loops or just to generate a numbered list.

```
list(range(10)) for i in range(10):  
                 print(i)
```

Dictionary

The Python dictionary stores data in terms of key:value pairs. A key can be any value of any data type except a list, another dictionary (other mutable types). The key is used to map to a specific value. A value can be of any data type similar to the element of a list.

```
{"name":"Juan Dela Cruz","age":2,"is_enrolled":False}  
{0:1, -1:2 ,2.0:3}
```

For more information you may also visit the official python documentation:

<https://docs.python.org/3.7/library/stdtypes.html#sequence-types-list-tuple-range>

<https://docs.python.org/3.7/library/stdtypes.html#mapping-types-dict>

4. Materials and Equipment:

Desktop Computer with Anaconda Python
Windows Operating System

5. Procedure:

Lists

1. Create a variable **numberlist** and assign it the value of [5,4,2,1,3].
2. Print the following values below:
 - a. len(numberlist)
 - b. numberlist[0]
 - c. numberlist[1]
 - d. numberlist[2]
 - e. numberlist[3]
 - f. numberlist[4]
 - g. numberlist[5]
 - h. numberlist[-1]
 - i. numberlist[-2]
 - j. numberlist[-3]
 - k. numberlist[-4]
 - l. numberlist[-5]
 - m. numberlist[-6]

Reminder: Use the print() command. The values numberlist[5] and numberlist[-6] should return an error.

3. Write your observation after printing all the values.
4. Create a variable named **itemlist** and assign it the value of [1,-2.0,[1,2,3], "Word"]
5. Print the following values below:
 - a. len(itemlist)
 - b. itemlist [0]
 - c. itemlist [1]
 - d. itemlist [2]
 - e. itemlist [3]
 - f. len(itemlist[2])
 - g. itemlist [2][0]
 - h. itemlist [2][1]
 - i. itemlist [2][2]
 - j. itemlist [-1]

- k. `itemlist [-2]`
- l. `itemlist [-3]`
- m. `itemlist [-4]`
- n. `len(itemlist[-2])`
- o. `itemlist[-2][0]`
- p. `itemlist[-2][1]`
- q. `itemlist[-2][2]`
- r. `itemlist[-2][-3]`
- s. `itemlist[-2][-2]`
- t. `itemlist[-2][-1]`

6. Write your observation after printing all the values. What does `len()` do?

Index Slicing

1. Create a new variable **longlist** and assign it the value of `numberlist + itemlist`.
2. Print the following values below and write your observation for each of the following sub-groups (sub-headings):
 - a. `len(longlist)`
 - b. `longlist [:]`
 - c. `longlist[:9]`
 - d. `longlist[0:]`
 - e. `longlist[1:]`
 - f. `longlist[2:]`

Index Slicing with Range

- g. `longlist[2:5]`
- h. `longlist[5:2]`
- i. `longlist[8:]`
- j. `longlist[9:]`

Index Slicing using Negative Indices

- k. `longlist[-9:]`
- l. `longlist[-8:]`
- m. `longlist[-8:-7]`
- n. `longlist[-1:]`

Other properties of Index Slicing

- o. `longlist[10:20]`
- p. `longlist[-7:5]`

Index Slicing with Step parameter

- q. `longlist[::1]`
- r. `longlist[::2]`
- s. `longlist[1:8:2]`
- t. `longlist[9:1:-1]`
- u. `longlist[-1::1]`
- v. `longlist[-1::-1]`

3. Write your main observation about index slicing as a whole.

List Methods and the Mutable Property of Lists

1. Create a new variable **numberlist2** and assign it to be equal to **numberlist**.
2. Print the value of **numberlist**.
3. Print the value of **numberlist2**.
4. Assign the value of **numberlist[0]** to be equal to 6.
5. Print the value of **numberlist**.
6. Print the value of **numberlist2**.

7. Observe how **numberlist2** is affected by changes in **numberlist** due to the assignment.
8. Change the value of **numberlist2** and assign it the value of **numberlist.copy()**
9. Print the value of **numberlist2**
10. Assign the value of **numberlist[0]** to be equal to 5.
11. Print the value of **numberlist**.
12. Print the value of **numberlist2**.
13. Write your observation about the immutable property and the difference of assigning **numberlist2** to be equal to **numberlist** and the **numberlist.copy()** method.

Exploring some List Functions and Methods

1. Print the value of **numberlist**
2. Run the command **numberlist.append(6)**
3. Print the value of **numberlist**
4. Run the command **numberlist.pop()**
5. Print the value of **numberlist**
6. Run the command **numberlist.sort()**
7. Print the value of **numberlist**
8. Run the command **itemlist.sort()**
9. Print the values: **min(numberlist)** and **max(numberlist)**
10. Print the value of **longlist**
11. Print the value of **longlist.count(1)**
12. Print the value of **longlist[7].count(1)**

The in operator

1. Type the code as shown: **print(3 in longlist)**
2. Type the code as shown: **print(15 in longlist)**
3. Type the code as shown below:

```
num = int(input("Enter a number: "))  
if num in longlist:  
    print("The number is in longlist")  
else:  
    print("The number is not in longlist")
```
4. Write your observations on the **in** operator.

Using a list in an iterative statement

1. Type the code as shown below:

```
for item in longlist:  
    print(item)
```
2. Type the code as shown below:

```
i=0  
while i<len(longlist):  
    print(longlist[i])  
    i+=1
```

Strings

1. Create a variable named **message** and assign it the value of "Hello World"
2. Print the value of **message**
3. Print the value: **len(message)**
4. Apply the concept of index values in the **List** section and individually display the characters "H", "E", "L", "L", "O" using the **print()** function.

Note: Try using positive indexes, then after seeing the result. Repeat the step using negative indexes.

5. Apply the concept of index values in the **List** section and display the string "Hold" using the Concatenate (+) operator on individual characters.
Ex. `print(message[0]+ message[1]+ message[2]+ message[3]+ message[4])`
6. Apply the concept of index slicing in the **Index Slicing** section and display the word "Hello" as a whole string.
7. Apply the concept of index slicing in the **Index Slicing** section and display the word "World" as a whole string.

String Methods

Observe the result per each String method.

1. Type the command and print the value `message.upper()`
Ex. `print(message.upper())`
2. Type the command and print the value `message.lower()`
3. Type the command and print the value `message.title()`
4. Print the value "Value 1 is {}, and value 2 is {}".format(-1,True)
5. Print the value `message.split(' ')`
6. Print the value `message.count('l')`
7. Print the value `message.replace('World','CPE009')`
8. Assign the value `message.replace('World','CPE009')` to `message`
9. Type the command: `help("")`
Find the commands used in previous tasks.

The in operator for Strings

1. Type the code as shown: `print('W' in message)`
2. Type the code as shown: `print('old' in message)`
3. Type the codes below:
`word = input("Enter a word: ")`
`if word in "The big brown fox jump over the lazy dog":`
`print("The word is in the text")`
`else:`
`print("The word is not in the text")`

Using a String in an iterative statement

1. Type the code as shown below:
`for character in message:`
`print(character)`
2. Type the code as shown below:
`i = 0`
`while i<len(message):`
`print(message[i])`
`i+=1`

Tuples

1. Create a variable named `tuplelist` and assign the value of (1,2,3,4,5)
2. Print the following values:
 - a. `numberlist[0]`
 - b. `numberlist[1]`
 - c. `numberlist[2]`
 - d. `numberlist[3]`
 - e. `numberlist[4]`
 - f. `numberlist[5]`

3. Print the output of tuplelist + (1,2,3)
4. Assign tuplelist[0] = 15
5. Observe the output.
6. Try string slicing through the elements of tuplelist as in numberlist and message.
7. Create a for loop that would print the numbers inside the tuple.

Dictionaries

1. Create a dictionary named **contactinfo** = {'id':1, 'first_name':'John', 'last_name':'Doe', 'contact_number':'09060611233'}
2. Print the following values:
 - a. contactinfo['id']
 - b. contactinfo['first_name']
 - c. contactinfo['last_name']
 - d. contactinfo['contact_number']
 - e. contactinfo['age']
3. Type the code:
for k,v in contactinfo:
 print(k)
4. Type the code:
for k,v in contactinfo.items():
 print(k,v)
5. Assign the values:
 - a. contactinfo['id'] = 2
 - b. contactinfo['first_name'] = 'Max'
6. Print **contactinfo**

6. Supplementary Activity:

Tasks

Distance Formula

1. Make a program that would calculate the distance between two points given a list of coordinates. Use the distance formula.

coordinates_list = [(1,1), (2,3)]

Simple Word Filter

2. For a given string input, replace all the words "stupid" with an asterisk * equal to the length of the string. The new string value should be displayed with the asterisks.

Phonebook

3. Create a simple phonebook program that can read from a list of dictionaries. The program should be able to display a person's name, contact, and address based from a user input which is the id of the record.

Output: Please refer to this link: [Laboratory No.4 - Colab](#)

Questions

1. How do we display elements of lists, tuples, and strings?
 - Access elements of lists, tuples and strings by indexing using indexing point (starting from 0) or loop through them to display all elements. Lists and tuples contain elements while strings store characters all accessible by their position.

2. What is the difference between a list, tuple, string and dictionary? Give possible use case for each.
- Lists are changeable, ordered, and store items like to-do lists or names. Tuples are unchangeable, ordered, and ideal for fixed data like coordinates. Strings are unchangeable, ordered text for names or sentences. Dictionaries are changeable, store key-value pairs like phonebooks, and allow access by key. Choose the data structure that fits your needs for modifying data, order, and key-based access.

3. Discuss the various string methods that were used in the activity. What do each of the methods do?
- The activity used string methods to manipulate text: changing case with upper/lower, capitalizing with title, inserting values using format, splitting into words with split, counting substrings with count, and replacing text with replacement. These versatile tools enable various string operations in Python.

8. Conclusion:

- This discussion covered Python's core data structures—lists, tuples, strings, and dictionaries—along with their properties and applications. Accessing elements by index or looping and manipulating strings with methods like upper, split, and replace were demonstrated. Mastering these data structures and string methods is essential for effective data management and text processing in Python. Choosing the right data structure and utilizing these tools empower programmers to achieve desired results in their code.

8. Assessment Rubric: