Laboratory Activity No. 4	
Sequence and Mapping Types	
Course Code: CPE103	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed: FEB. 8, 2025**
Section: 1A	**Date Submitted: FEB. 15, 2025**
Name: CATAHAN, JOSHUA A.	**Instructor: ENGR. MARIA RIZETTE 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:

PLEASE PROCEED TO THIS LINK TO CHECK ALL THE CODED TASKS:

https://colab.research.google.com/drive/1aPmYmFQXR4AIXHfccCPaRFEazt7BBBJL#scrollTo=pJAYeRMSyDX-&line=1&uniqifier=1

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.

When I printed the values in numberlist, everything worked fine for valid indices, like numberlist[0] to numberlist[4]. But when I tried to access numberlist[5] or numberlist[-6], I got an error because there aren't that many elements in the list. So, trying to access elements that don't exist causes an error.

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?

The len() function worked well to tell me how many items were in itemlist, even though it had different types of values, like numbers and a list inside it. It counts all the elements at the top level, and it helps me quickly check how many items are in a list or sublist.

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:]

OBSERVATION:

Index slicing is a super handy way to get specific parts of a list. I can use it to grab a section of the list, like longlist[1:3], or even just copy the whole list with longlist[:]. Negative indices help me count from the end of the list, and the step parameter lets me skip over elements, like longlist[::2] to pick every second item.

Index Slicing with Range

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

OBSERVATION (Index Slicing with Range):

When I slice using a range, I can pick a specific part of the list by defining a start and end, like longlist[2:5]. If the range doesn't make sense, like longlist[5:2], it just gives me an empty list. Using steps in a range, like longlist[1:8:2], helps me get elements in a specific order or pattern.

Index Slicing using Negative Indices

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

OBSERVATION (Index Slicing using Negative Indices):

Negative indices let me count from the end of the list, which is really useful. For example, longlist[-1:] gives me the last item, and longlist[-8:-7] grabs just one item from the end. I can mix negative indices with other slicing techniques to easily grab parts of a list in reverse.

Other properties of Index Slicing

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

OBSERVATION (Other properties of Index Slicing):

Sometimes, when I try to slice beyond the list's length, like longlist[10:20], I'll get an empty list because there aren't enough elements. It's important to remember that if the slice doesn't match the list's size, Python will just return nothing.

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]

OBSERVATION (Index Slicing with Step parameter):
Using the step parameter in slicing is great for skipping items in a list. For example, longlist[::2] grabs every second item. If I want to reverse the order of a list, I can use negative steps, like longlist[9:1:-1]. It gives me lots of flexibility to rearrange or selectively pick elements.

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

Overall, index slicing is a powerful and flexible tool. It lets me pick exactly what I want from a list, whether that's a range of elements, every other item, or even reversing the order. It's super useful for manipulating lists without needing loops or complex code.

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.

When I did numberlist2 = numberlist, both variables pointed to the same list. So, when I changed numberlist, numberlist2 changed too, since they both referred to the same list in memory. This is something to keep in mind when working with lists—you're not always copying the list, just referencing it.

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.

When I assigned numberlist2 directly to numberlist, both variables were linked to the same list, so any change to one affected the other. But when I used .copy(), numberlist2 became a completely separate list. This showed me that .copy() makes a true copy of the list, so changes to one list won't affect the other.

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.

The in operator was a fast and easy way to check if an item is in a list. It returned True if the item was found, and False if it wasn't. It's great for checking membership in lists or strings.

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())
   The upper() method converts all characters in the string to uppercase.
2. Type the command and print the value message.lower()
   The lower() method converts all characters in the string to lowercase
3. Type the command and print the value message.title()
   The title() method capitalizes the first letter of each word in the string.
4. Print the value "Value 1 is {}, and value 2 is {}".format(-1,True)
   The format() method allows you to insert values into a string using curly braces {} as placeholders.
5. Print the value message.split(' ')
   The split() method splits a string into a list of words based on a delimiter. In this case, the delimiter is a space ' '.
6. Print the value message.count('l')
   The count() method counts how many times a substring occurs in the string.
7. Print the value message.replace('World','CPE009')
   The replace() method replaces a substring with another substring.
8. Assign the value message.replace('World','CPE009') to message
   The message would return message.replace('World','CPE009')
9. Type the command: help("")
   Find the commands used in previous tasks.
   The help() function provides a helpful summary of a Python object's methods and functions.

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]
    g. Print the output of tuplelist + (1,2,3)
    h. Assign tuplelist[0] = 15
    i. Observe the output.
    When I tried to modify tuplelist, I got an error because tuples are immutable. This means once they're created, you can't change their values. It's a good feature when you want to make sure the data can't be accidentally modified.
    j. Try string slicing through the elements of tuplelist as in numberlist and message.
    k. 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'
   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.

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

LINK: https://colab.research.google.com/drive/1aPmYmFQXR4AIXHfccCPaRFEazt7BBBJL#scrollTo=jy8zwLtvKMuB&line=2&uniqifier=1

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.

LINK: https://colab.research.google.com/drive/1aPmYmFQXR4AIXHfccCPaRFEazt7BBBJL#scrollTo=Qoz1-vrOKu0Q&line=2&uniqifier=1

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.

LINK: https://colab.research.google.com/drive/1aPmYmFQXR4AIXHfccCPaRFEazt7BBBJL#scrollTo=81RrrOufKw6p&line=3&uniqifier=1

Questions

1. How do we display elements of lists, tuples, and strings?

We can display the elements by using their index or by looping through them one by one.

2. What is the difference between a list, tuple, string and dictionary? Give possible use case for each.

A list can be changed and is used when we need to store things that might change, like a to-do list. A tuple can't be changed and is used when we need a fixed set of items, like coordinates. A string is used to store text, like a name or message. A dictionary stores data in key-value pairs, like storing contact information with names as keys.

3. Discuss the various string methods that were used in the activity. What does each of the methods do?

The activity used several string methods: upper() converts the string to uppercase, lower() makes it lowercase, and title() capitalizes the first letter of each word. The split() method breaks the string into words, count() counts occurrences of a substring, and replace() replaces one substring with another. These methods simplify text manipulation.

8. Conclusion:

In this task, we learned how to work with lists, tuples, and dictionaries in Python. Lists are flexible and can be changed, while tuples are fixed once created. We used index slicing to easily select and manipulate parts of lists. We also explored string methods to modify and access parts of text. The in operator helped us check for elements in both lists and strings. These concepts will help you manage and work with data more efficiently in Python. By understanding these data types and techniques, you can store, access, and manipulate data in various ways depending on your needs. Index slicing makes it simple to extract or modify parts of lists, while string methods help you manipulate text easily. The in operator allows you to quickly check if an element exists in a collection. Overall, mastering these tools will make your code cleaner, more flexible, and easier to work with.

8. Assessment Rubric: