University Of Engineering and Technology, Lahore
Computer Engineering Department

Course Name: Fundamentals of Programming & Data Science	**Course Code:** CMPE-112L
Assignment Type: Lab	**Dated:** 19th February 2024
Semester: 2nd	**Session:** 2023
Lab/Project/Assignment #: 5	**CLOs to be covered: CLO2**
Lab Title: Set and Dictionary Data type	**Teacher Name:** Engr. Afeef Obaid

Lab Evaluation:

CLO2	Practice collaboratively on large problems and provide their working solutions.					
Levels (Marks)	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**	**Level 6**
(10)						
					Total	**/10**

Rubrics for Current Lab Evaluation

Scale	**Marks**	**Level**	**Rubric**
Excellent	**9-10**	L1	Submitted all lab tasks, BONUS task, have good understanding.
Very Good	**7-8**	L2	Submitted the lab tasks but have good understanding
Good	**5-6**	L3	Submitted the lab tasks but have weak understanding.
Basic	**3-4**	L4	Submitted the lab tasks but have no understanding.
Barely Acceptable	**1-2**	L5	Submitted only one lab task.
Not Acceptable	**0**	L6	Did not attempt

LAB No. 5

Lab Goals/Objectives:

By reading this manual, students will be able:

- To understand the concept of Set data type in Python
- To Become familiar with Set methods in Python
- To understand the concept of Dictionary data type in Python
- To Become familiar with Dictionary Methods in Python

Equipment Required: Computer system with Pycharm IDE and python package installed on it

Python Set

- In Python, a Set is a collection of items which are unordered, unchangeable, unindexed and contain no duplicate items.
- To create a Set in Python, we enclose a comma-separated sequence of items within curly brackets { } or we use type casting to convert list into set by using set(list).

  <u>Example</u>

  ```
  s={1,3.6,"Uet"}
  s1=set([44.5,"UET",7])
  print(s)
  print(s1)
  print(type(s))
  print(type(s1))
  ```

- Unordered means that the items in a set do not have a defined order.

  Set items can appear in a different order every time you use them, and cannot be referred to by index or key

- Unchangeable means that we cannot change the items after the set has been created.

  Once a set is created, you cannot change its items, but you can remove items and add new items

- Duplicates are not allowed means that sets cannot have two items with the same value

  <u>Example</u>

  ```
  s={1,3.6,"Uet",'CE',"Uet"}
  print(s)
  ```

  Note: The values True and 1 are considered the same value in sets, and are treated as duplicates

  ```
  s={1,3.6,"Uet",'CE',"Uet",1,True,1}
  print(s)
  ```

- In Python, an empty set is created by set() keyword, { } is not an empty set.

<u>Example</u>

```
s1={}
s=set()
print(type(s))
print(type(s1))
```

- Access Set Items

You cannot access items in a set by referring to an index or a key.

But you can loop through the set items using a for loop

<u>Example</u>

```
s1={"CE","CS","EE","MME","CE"}
for dept in s1:
   print(dept)
```

You can check if a specified value is present in a set, by using the in keyword

<u>Example</u>

```
s1={"CE","CS","EE","MME","CE"}
if "CE" in s1:
   print("CE exist in set")
```

Set Methods

- **add()**

The add() method adds an element to the set.

If the element already exists, the add() method does not add the element

<u>Syntax</u>

```
set.add(element)
```

Example

```
s1={"CE","CS","EE","MME","CE"}
s1.add("SE")
s1.add("EE")
print(s1)
```

- **discard()**

The discard() method removes the specified item from the set.

Syntax

```
set.discard(element)
```

Example

```
s1={"CE","CS","EE","MME","CE"}
s1.discard("SE")
s1.discard("EE")
print(s1)
```

- **remove()**

The remove() method removes the specified element from the set.
This method is different from the discard() method, because the remove() method will raise an error if the specified item does not exist, and the discard() method will not

Syntax

```
set.remove(element)
```

Example

```
s1={"CE","CS","EE","MME","CE"}
s1. remove("SE")
s1. remove("EE")
print(s1)
```

- **pop()**

The pop() method removes a random item from the set.
This method returns the removed item.

Syntax

set.pop()

Example

```
s1={"CE","CS","EE","MME","CE"}
remove=s1.pop()
print(remove)
print(s1)
```

- **clear()**

The clear() method removes all elements in a set.

Syntax

set.clear()

Example

```
s1={"CE","CS","EE","MME","CE"}
s1.clear()
print(s1)
```

- **union()**

The union() method returns a set that contains all items from the original set, and all items from the specified set(s). You can specify as many sets you want, separated by commas.
If an item is present in more than one set, the result will contain only one appearance of this item.

Syntax

set.union(set1, set2...)

Example

```
s1={"CE","CS","EE","MME","CE"}
s2={"PID","CRP","EE"}
print(s1.union(s2))
print(s1,s2)
```

- **update()**

The update() method updates the current set, by adding items from another set
If an item is present in both sets, only one appearance of this item will be present in the updated set.

Syntax

```
set.update(set)
```

Example

```
s1={"CE","CS","EE","MME","CE"}
s2={"PID","CRP","EE"}
s1.update(s2)
print(s1,s2)
```

- **intersection()**

The intersection() method returns a set that contains the similarity between two or more sets.

Syntax

```
set.intersection(set1, set2 ... etc)
```

Example

```
s1={"CE","CS","EE","MME","CE"}

s2={"PID","CRP","EE"}

print(s1.intersection(s2))

print(s1,s2)
```

- **intersection_update()**

The intersection_update() method is different from the intersection() method, because the intersection() method returns a new set, without the unwanted items, and the intersection_update() method removes the unwanted items from the original set.

<u>Syntax</u>

set.intersection_update(set1, set2 ... etc)

<u>Example</u>

```
s1={"CE","CS","EE","MME","CE"}
s2={"PID","CRP","EE"}
s1.intersection_update(s2)
print(s1,s2)
```

- **symmetric_difference()**

The symmetric_difference() method returns a set that contains all items from both set, but not the items that are present in both sets.

<u>Syntax</u>

set.symmetric_difference(set)

<u>Example</u>

```
s1={"CE","CS","EE","MME","CE"}
s2={"PID","CRP","EE"}
print(s1.symmetric_difference(s2))
print(s1,s2)
```

- **difference()**

The difference() method returns a set that contains the difference between two sets
The returned set contains items that exist only in the first set, and not in both sets

<u>Syntax</u>

set.difference(set)

Example

```
s1={"CE","CS","EE","MME","CE"}
s2={"PID","CRP","EE"}
print(s1.difference(s2))
print(s1,s2)
```

- **isdisjoint()**

The isdisjoint() method returns True if none of the items are present in both sets, otherwise it returns False.

Syntax

```
set.isdisjoint(set)
```

Example

```
s1={"CE","CS","EE","MME","CE"}
s2={"PID","CRP","EE"}
print(s1.isdisjoint(s2))
```

- **issubset()**

The issubset() method returns True if all items in the set exists in the specified set, otherwise it returns False

Syntax

```
set.issubset(set)
```

Example

```
s1={"CE","CS","EE","MME","CE"}
s2={"PID","CRP","EE"}
print(s1.issubset(s2))
```

- **issuperset()**

The issuperset() method returns True if all items in the specified set exists in the original set, otherwise it returns False.

Syntax

```
set.issuperset(set)
```

Example

```
s1={"CE","PID","EE","MME","CRP"}
s2={"PID","CRP","EE"}
print(s1.issuperset(s2))
```

Python Dictionaries

- Dictionary items are ordered, changeable, and does not allow duplicates.
- Dictionary items are presented in key:value pairs, and can be referred to by using the key name.
- The values in dictionary items can be of any data type.

Example

```
marks={
  "I2C":70, "PF":80, "CA":75
}
```

- To determine how many items a dictionary has, use the len() function and to check the data type of dictionary use type() function

Example

```
print(len(marks))
print(type(marks))
```

- You can access the items of a dictionary by referring to its key name, inside square brackets, There is also a method called get() that will give you the same result but the difference is if key doesn't match get() method does not throw error whereas the first method display error.

<u>Example</u>

```
print(marks["CA"])
print(marks.get("CA"))
print(marks.get("Cs"))
print(marks["Cs"])
```

Dictionary Methods

- The keys() method will return a list of all the keys in the dictionary.

  <u>Syntax</u>

  ```
  dictionary.keys()
  ```

  <u>Example</u>

  ```
  print(marks.keys())
  ```

- The values() method will return a list of all the values in the dictionary.

  <u>Syntax</u>

  ```
  dictionary.values()
  ```

  <u>Example</u>

  ```
  print(marks.values())
  ```

- The items() method will return each item in a dictionary, as tuples in a list.

  <u>Syntax</u>

  ```
  dictionary.items()
  ```

Example

```
print(marks.items())
```

- The pop() method removes the specified item from the dictionary.

  Syntax

  ```
  dictionary.pop(keyname, defaultvalue)
  ```

  Example

  ```
  x=marks.pop("I2C")
  print(x,marks)
  x=marks.pop("IBC",-1)
  print(x,marks)
  x=marks.pop("IBC")
  ```

- The popitem() method removes the item that was last inserted into the dictionary.

  Syntax

  ```
  dictionary.popitem()
  ```

  Example

  ```
  x=marks.popitem()
  print(x,marks)
  ```

- The clear() method removes all the elements from a dictionary.

  Syntax

  ```
  dictionary.clear()
  ```

  Example

  ```
  marks.clear()
  print(marks)
  ```

- The copy() method returns a copy of the specified dictionary.

  Syntax

  dictionary.copy()

  Example

```
m=marks.copy()
print(marks,m)
```

- The update() method inserts the specified items to the dictionary.

  Syntax

  dictionary.update({key:value})

  Example

```
marks={
   "I2C":70, "PF":80, "CA":75
}
marks1={
   "OOP":90, "Java":40
}
marks.update(marks1)
print(marks)
```

Check Item exist in Dictionary

- To determine if a specified key is present in a dictionary use the **`in`** keyword

  Example

```
if "PF" in marks:
   print("Yes")
```

Access Dictionary Items

- You can loop through a dictionary by using a for loop.
- When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

Example

```
for sub in marks:
   print(sub)
for sub in marks.keys():
   print(sub)
```

- Print all values in the dictionary, one by one

Example

```
for sub in marks:
   print(marks[sub])
print("=========")
for mark in marks.values():
   print(mark)
```

- Loop through both keys and values, by using the items() method:

Example

```
for sub,mark in marks.items():
   print(sub,mark)
```

Nested Dictionaries

- A dictionary can contain dictionaries, this is called nested dictionaries.

Example

```
marks1={
   "I2C":70, "PF":80, "CA":75
}
marks2={
   "I2C":79, "PF":90, "CA":86
}
marks={
   "student1":marks1,
   "student2":marks2
}
print(marks["student1"]["PF"])
```

Lab Tasks

- You have two datasets containing the names of candidates who applied for Engineering and Non engineering program in UET. Write the python program to display the name of candidates who applied in both programs and total number of candidates who applied in UET.

- Write the python program for the Phonebook which has the following features
    1. Search Contact number by name
    2. Display names of all contacts
    3. Display all contact numbers
    4. Delete any contact number
    5. Update any contact number
    6. Delete complete phonebook

- Write the python program that takes a paragraph from the user and display total number of words in the paragraph and also display how many times a same word repeats in the paragraph by using dictionary.