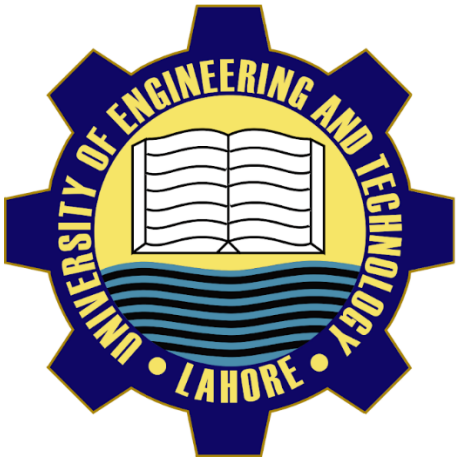
MCT-242 : COMPUTER PROGRAMMING-I

using Python 3.9



Prepared By:

Mr. Muhammad Ahsan Naeem

YouTube Playlist

https://www.youtube.com/playlist?list=PLWF9TXck7O_wMDB-VriREZ6EvwkWLNB7q

LAB 23: SET

Yet another data type in Python is `Set` equivalent to **Set Theory** in mathematics that is a collection of unorder elements. The elements of set must be **unique** and of **immutable** data type. A set by itself is mutable i.e. we can add or remove elements from a set but the individual elements in set must be immutable. Mathematical set operations like union, intersection, symmetric difference etc. can be performed on these.

A set is defined and printed here:

```
A={1,3,5,6}
print(A)
```

If we try to put multiple values in set, it will not generate an error but will remove the duplicates.

```
A={1,6,3,5,6}
print(A)
```

The elements in python sets are unique; there can't be duplicate items in python sets. If duplicate items entered, it will be ignored, and final set will always contain unique elements. The elements of the set cannot be accessed using indexes. There is no index attached to set items.

Sets are implemented in Python using **Hash Table** to store the elements and the order depends on the **Hash Function** used there.

Set is iterable so we can apply the `for` loop directly on the set. Moreover, we can use the `in` operator for membership test on set.

Empty Set:

A set without any element is an empty set. Let's try to define an empty set as:

```
a={}
print(type(a))
```

The output is:

```
<class 'dict'>
```

Which means, `a` is a dictionary and not a set. Hence, we cannot use the above method to define the empty set. Instead, we will have to use the set constructor as:

```
a=set()
print(type(a))
```

Now the output is:

```
<class 'set'>
```

We cannot access individual elements inside a set using the index as in case of list and tuple. But we can always add or remove elements. The methods available in Set class are given here:

Method	Description
`add()`	Adds one element into the set.
`clear()`	Removes all elements from the set
`copy()`	Returns a copy of the set
`difference()`	Returns the difference of two or more sets as a new set
`difference_update()`	Removes all elements of another set from this set
`discard()`	Removes an element from the set if it is a member. (Do nothing if the element is not in set)
`intersection()`	Returns the intersection of two sets as a new set
`intersection_update()`	Updates the set with the intersection of itself and another
`isdisjoint()`	Returns True if two sets have a null intersection
`issubset()`	Returns True if another set contains this set
`issuperset()`	Returns True if this set contains another set
`pop()`	Removes and returns an arbitrary set element. Raise **KeyError** if the set is empty
`remove()`	Removes an element from the set. If the element is not a member, raise a **KeyError**
`symmetric_difference()`	Returns the symmetric difference of two sets as a new set
`symmetric_difference_update()`	Updates a set with the symmetric difference of itself and another
`union()`	Returns the union of sets in a new set
`update()`	Updates the set with the union of itself and others

The different operators which can be used on two sets are given here:

Operator	Description
|	Union. Creates new set
&	Intersection. Creates new set.
-	Difference. Creates new set.
^	Symmetric Difference. create a new set containing elements from both the sets except common elements.
==	Equality Test
<	Subset Test
>	Superset Test

The above operators are applied on two sets. But different set methods, for example `union()` method, will take any iterable as an argument, convert it to a set, and then perform the union.

Example Use cases of Sets:

We did a task of setting a valid password which must have at least one upper case letter, one lower case letter, one number and one special character. This can be done as:

```
import string
pwd=input('Set you password:')

check1=check2=check3=check4=False
for char in pwd:
    if(char.isupper()):
        check1=True
    elif(char.islower()):
        check2=True
    elif(char.isdecimal()):
        check3=True
    elif(char in string.punctuation):
        check4=True
if(check1 and check2 and check3 and check4):
    print('Password set correctly!')
else:
    print('Your password is NOT strong!')
```

In above case we are iterating over each character of the password to check the conditions. Instead, we could do it as:

```
import string
pwd=input('Set your password:')
valid=True
if(set(pwd).intersection(string.ascii_lowercase)==set()):
    valid=False
elif(set(pwd).intersection(string.ascii_uppercase)==set()):
    valid=False
elif(set(pwd).intersection(string.digits)==set()):
    valid=False
elif(set(pwd).intersection(string.punctuation)==set()):
    valid=False
if(valid):
    print('Password set correctly!')
else:
    print('Your password is NOT strong!')
```

Now suppose that we have a **5x5 grid world** (total 25 cells). There are two wheeled robots which can scan the grid using different sensors attached to those. The two robots are placed on two corners as shown below:

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

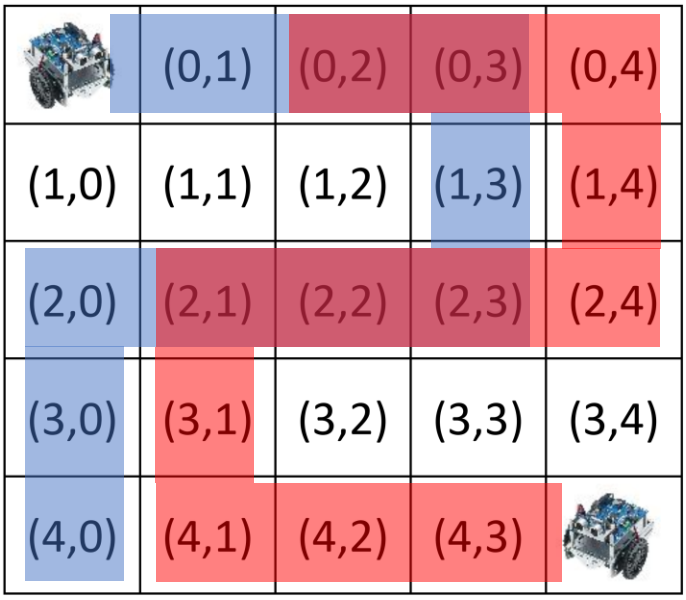
Suppose that one robot will scan the grid and it will pass the information of the scanned cells to a controller where a cell is defined as row number and column number. The path for first robot is shown below:

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

Likewise, second robot will also scan the grid and its path is shown here:

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

You can see there are some cells which are scanned by both Robots and we have a few cells which are not scanned by any of the robot. The combined scenario is shown below:



Now if we are interested to get the information of the cells scanned by both Robots and the cells which are left unscanned, we can better use set operations for that. We can use list for the grid and the path with a cell represented as a Tuple and to apply different set operations, we can covert those to sets using the set constructor. Later, we can change the result to list again.

```
grid=[(i,j) for i in range(5) for j in range(5)]
print(grid)
robo1=[(0,0),(0,1),(0,2),(0,3),(1,3),(2,3),(2,2),(2,1),(2,0),(3,0),(4
,0)]
robo2=[(4,4),(4,3),(4,2),(4,1),(3,1),(2,1),(2,2),(2,3),(2,4),(1,4),(0
,4),(0,3),(0,2)]
# unscannedCells=set(grid)-set(robo1)-set(robo2)
unscannedCells=set(grid).difference(robo1,robo2)
print(list(unscannedCells))
dualScanned=list(set(robo1).intersection(robo2))
print(dualScanned)
```

Set Comprehension:

Just like list comprehension, we have the set comprehension. For example, if we want to create a set of squares of all integers from 1 to 20, we can do it as:

```
s={i*i for i in range(20)}
print(s)
```

The different techniques we studied in List Comprehension are applicable in Set Comprehension.

Tasks:

[1] Define a List of numbers and then find and display the second largest number in the list. If there are repeating largest number in the List, the second largest is the one different from the largest.

[2] Suppose you have two lists names as `a` and `b` containing some numbers. You need another list as common elements between the lists `a` and `b` except the first and the last number of the list `a`.