List, Tuple, Set and Dictionary in Python

Problem 01

Write a Program that creates a list of squares upto to 10 numbers.

Simple Numbers

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

Squares of the Numbers

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Solution with Traditional for Loop

Write a Program that creates a list of squares upto to 10 numbers.

```
squares = []
    for x in range(10):
        squares.append(x**2)
    print(squares)
```

Problem 01

Write a Program that creates a list of squares upto to 10 numbers.

Is there any more concise and readable Solution in Python?

Solution with Comprehension

Write a Program that creates a list of squares upto to 10 numbers.

```
squares = [x**2 for x in range(10)]
    print(squares)
```

Problem 02

Write a program in Python that combines the elements of two lists if they are not equal.

For Example

```
List1 = [1,2,3]
List2 = [3,1,4]
```

Output:

```
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Solution with Traditional for loop

Write a program in Python that combines the elements of two lists if they are not equal.

```
combs = []
    for x in [1,2,3]:
     for y in [3,1,4]:
         if x != y:
             combs.append((x, y))
    print(combs)
```

Solution with Comprehension

Write a program in Python that combines the elements of two lists if they are not equal.

```
combs = [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
    print(combs)
```

Problem 03

Write a program to access the grade of a student by providing the name of the student.

Problem 03

Write a program to access the grade of a student by providing the name of the student.

Now, we have to decide how to store that data that will be beneficial for us in case of accessing and searching.

Solution with Dictionary

Dictionary is an unordered collection of key-value pairs. Each entry has a key and value. A dictionary can be considered as a list with special index.

Solution with Dictionary

We can create a dictionary by providing 0 or more key value pairs between curly braces.

```
grades = {'John':'A', 'Emily':'A+', 'Betty':'B', 'Mike':'C', 'Ashley':'A'}
```

Solution with Dictionary

We can create a dictionary by providing 0 or more key value pairs between curly braces.

grades = {'John':'A', 'Emily':'A+', 'Betty':'B', 'Mike':'C', 'Ashley':'A'}

```
grades = {'John':'A', 'Emily':'A+', 'Betty':'B', 'Mike':'C', 'Ashley':'A'}
print(grades['John'])
```

List, Tuple, Set and Dictionary in Python

Storing Data in Python

Python Collections are used to store data, for example, lists (arrays), dictionaries, sets, and tuples.

List	Tuple	Set	Dictionary
A list is a collection of *ordered* data.	A tuple is an *ordered* collection of data.	A set is an *unordered* collection.	A dictionary is an *unordered* collection of data that stores data in key-value pairs.

Storing Data in Python

Python Collections are used to store data, for example, lists (arrays), dictionaries, sets, and tuples.

List	Tuple	Set	Dictionary
A list is a collection of *ordered* data.	A tuple is an *ordered* collection of data.	A set is an *unordered* collection.	A dictionary is an *unordered* collection of data that stores data in key-value pairs.
Lists are declared with square braces.	Tuples are enclosed within parentheses.	Sets are represented in curly brackets.	Dictionaries are enclosed in curly brackets in the form of key-value pairs.

Creating List, Tuple, Set and Dictionary

List

```
list1=[1,4,"Gitam",6,"college"]
list2=[]  # creates an empty list
list3=list((1,2,3))
```

Tuple

```
tuple1=(1,2,"college",9)
tuple2=() # creates an empty tuple
tuple3=tuple((1,3,5,9,"hello"))
```

Set

```
set1={1,2,3,4,5,"hello","tup"}
set2={(1,8,"python",7)}
```

Dictionary

```
dict1={"key1":"value1","key2":"value2"}
dict2={}    # empty dictionary
dict3=dict({1:"apple",2:"cherry",3:"strawberry"})
```

Accessing and Updating Data

An object is mutable if the value can change; else, the object is immutable.

List	Tuple	Set	Dictionary
Lists are mutable.	Tuples are immutable.	Sets are mutable and have no duplicate elements.	Dictionaries are mutable and keys do not allow duplicates.

Accessing List, Tuple, Set and Dictionary

List

```
list1=["hello",1,4,8,"good"]
print(list1)
list1[0]="morning"
# assigning values ("hello" is replaced with "morning")
print(list1)
print(list1[4])
print(list1[-1])
# list also allow negative indexing
print(list1[1:4]) # slicing
list2=["apple","mango","banana"]
print(list1+list2) # list concatenation
```

Accessing List, Tuple, Set and Dictionary

Tuple

```
tuple1=("good",1,2,3,"morning")
print(tuple1)
print(tuple1[0])  # accessing values using indexing
#tuple1[1]="change"  # a value cannot be changed as they are immutable
tuple2=("orange","grapes")
print(tuple1+tuple2)  # tuples can be concatenated
tuple3=(1,2,3)
print(type(tuple3))
```

Accessing List, Tuple, Set and Dictionary

Set

```
set1={1,2,3,4,5}
print(set1)
# set1[0]   sets are unordered, so it doesnot support indexing
set2={3,7,1,6,1} # sets doesnot allow duplicate values
print(set2)
```

Accessing List, Tuple, Set and Dictionary

Dictionary

```
dict1={"key1":1,"key2":"value2",3:"value3"}
print(dict1.keys())  # all the keys are printed
dict1.values() # all the values are printed
dict1["key1"]="replace_one"  # value assigned to key1 is replaced
print(dict1)
print(dict1["key2"])
```

Built-In Methods for List, Tuple, Set, Dictionary

An object is mutable if the value can change; else, the object is immutable.

List	Tuple	Set	Dictionary
The append() method adds a single item at the end of the list without modifying the original list.	An element cannot be added to the tuple as it is immutable.	The set add() method adds a given element to a set.	The update() method updates the dictionary with the specified key-value pairs
The pop() method removes the item at the given index from the list and returns it.	Tuples are immutable.	The pop() method removes a random item from the set.	The pop() method removes the specified item from the dictionary.

Built-In Methods for List

List

```
list1=["apple","banana","grapes"]
list1.append("strawberry")    # strawberry is added to
the list
print(list1)
list1.pop()  # removes the last element from the list
print(list1)
list1.pop()
print(list1)
```

Built-In Methods for Tuple

Tuple

```
tuple1=(1,2,3,4)
# tuple1.pop()     tuple cannot be modified
# tuple1.append()  tuple cannot be modified
print(tuple1)
```

Built-In Methods for Set

Set

```
set1={"water","air","food"}
set1.add("shelter")   # adds an element to the set at random position
print(set1)
set1.add("clothes")
print(set1)
set1.pop()  # removes random element from the set
print(set1)
```

Built-In Methods for Dictionary

Dictionary

```
dict1={"fruit1":"apple","fruit2":"banana","veg1":"tomato"}
dict1.update({"veg2":"brinjal"})
print(dict1)
dict1.update({"veg3":"chilli"})  # updates the dictionary at the end
print(dict1)
dict1.pop("veg2")
print(dict1)
```

Searching Methods for List, Tuple, Set, Dictionary

List	Tuple	Set	Dictionary
index() searches for a given element from the start of the list and returns the lowest index where the element appears.	Searches the tuple for a specified value and returns the position of where it was found.	The index of a particular element is not retrieved as they are unordered.	The get() method returns the value of the item with the specified key.

Searching Methods for List

List

```
list1=[1,5,3,9,"apple"]
print(list1.index(9)) # returns the index value of the particular element
list2=[2,7,8,7]
print(list2.index(7)) # returns the index value of the element at its first occurence
print(list2.index(7,2)) # returns the index value of the element from the particular
start position given
```

Searching Methods for Tuple

Tuple

```
tuple1=(1,3,6,7,9,10)
print(tuple1.index(6))    #prints 2
print(tuple1.index(9))    #prints 4
```

Searching Methods for Set

Set

```
set1={1,5,6,3,9}
# set1.index()  will throw an error as they are unordered
```

Searching Methods for Dictionary

Dictionary

```
dict1={"key1":"value1","key2":"value2"}
# dict1.index("key1") will throw an error
print(dict1.get("key1"))
```

Looping Techniques for List in Python

Iterating Techniques for List

Following are the most common ways in which we can iterate on a List in Python.

1. Using For Loop
2. Using For loop and range() function
3. Using While loop
4. Using list comprehension
5. Using enumerate() function

1. Using For Loop

```
list = [1, 2, 3, 4, 5]

# Iterating list using for loop
for i in list:
    print(i)
```

```
G:\OOP 2022\Week 14>FirstPythonCode.py
1
2
3
4
5
```

2. Using For loop and range() function

```
list = [1, 2, 3, 4, 5]
    # getting length of list using len() function
    length = len(list)

    # using for loop along with range() function
    for i in range(length):
        print(list[i])
```

```
G:\OOP 2022\Week 14>FirstPythonCode.py
1
2
3
4
5
```

3. Using While loop

```
list = [1, 2, 3, 4, 5]

# Getting length of list using len() function
    length = len(list)
    i = 0
    while i < length:
        print(list[i])
        i += 1
```

```
G:\OOP 2022\Week 14>FirstPythonCode.py
1
2
3
4
5
```

4. Using list comprehension

```
list = [1, 2, 3, 4, 5]

    # Iterating list using list comprehension
    [print(i) for i in list]
```

```
G:\OOP 2022\Week 14>FirstPythonCode.py
1
2
3
4
5
```

5. Using enumerate() function

```
list = [1, 3, 5, 7, 9]

# Using enumerate() function to iterate the list
    for i, val in enumerate(list):
        print (i, ",",val)
```

```
G:\OOP 2022\Week 14>FirstPythonCode.py
0 , 1
1 , 3
2 , 5
3 , 7
4 , 9
```

Looping Techniques for Set in Python

Iterating Techniques for Set

Following are the most common ways in which we can iterate on a Set in Python.

1. Using for loop
2. Enumerating over a set
3. Using iter with for loop
4. Converting the set to a list
5. Using comprehension
6. Iterating over two sets simultaneously using zip()

1. Using For Loop

```
my_set = {'london', 'new york', 'seattle', 'sydney','chicago'}
    for item in my_set:
        print(item)
```

```
G:\OOP 2022\Week 14>FirstPythonCode.py
sydney
seattle
london
chicago
new york
```

2. Enumerating over a set

```
my_set = {'london', 'new york', 'seattle', 'sydney','chicago'}
    for counter, item in enumerate(my_set):
        print(counter,":",item)
```

```
G:\OOP 2022\Week 14>FirstPythonCode.py
0 : chicago
1 : seattle
2 : new york
3 : sydney
4 : london
```

3. Using iter with for loop

```
my_set = {'london', 'new york', 'seattle', 'sydney','chicago'}
    for item in iter(my_set):
        print(item)
```

```
G:\OOP 2022\Week 14>FirstPythonCode.py
sydney
seattle
london
chicago
new york
```

4. Converting the set to a list

```
my_set = {'london', 'new york', 'seattle', 'sydney','chicago'}
    my_list = list(my_set)
    for i in range(0,len(my_list)):
        print(my_list[i])
```

```
G:\OOP 2022\Week 14>FirstPythonCode.py
sydney
seattle
london
chicago
new york
```

5. Using comprehension

```
my_set = {'london', 'new york', 'seattle', 'sydney','chicago'}
    [print(item) for item in my_set]
```

```
G:\OOP 2022\Week 14>FirstPythonCode.py
sydney
seattle
london
chicago
new york
```

6. Iterating over two sets simultaneously using zip()

```
set1 = {10, 20, 30, 40, 50}
    set2 = {100, 200, 300, 400, 500}
    for (i1, i2) in zip(set1, set2):
        print(i1, i2)
```

```
G:\OOP 2022\Week 14>FirstPythonCode.py
50 400
20 100
40 500
10 200
30 300
```

Looping Techniques for Dictionary in Python

Iterating Techniques for Dictionary

Following are the most common ways in which we can iterate on a Dictionary in Python.

1. Iterating Through Keys Directly
2. Iterating Through .items()
3. Iterating Through .values()

1. Iterating Through Keys Directly

```
a_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}
    for key in a_dict:
        print(key)
```

```
G:\OOP 2022\Week 14>FirstPythonCode.py
color
fruit
pet
```

2. Iterating Through .items()

```
a_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}
for item in a_dict.items():
    print(item)
```

```
G:\OOP 2022\Week 14>FirstPythonCode.py
('color', 'blue')
('fruit', 'apple')
('pet', 'dog')
```

3. Iterating Through .values()

```
a_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}
for value in a_dict.values():
    print(value)
```

```
G:\OOP 2022\Week 14>FirstPythonCode.py
blue
apple
dog
```