GROUP WORK PROJECT # 2 MScFE 600: FINANCIAL DATA
GROUP NUMBER: 846

FULL LEGAL NAME	LOCATION (COUNTRY)	EMAIL ADDRESS	MARK X FOR ANY NON-CONTRIBUTING MEMBER
Shailza Virmani	India	virmanishailza@gmail.com	
Harshil Sumra	India	harshilsumra1997@gmail.com	
Wisdom Yakanu	Ghana	ellinamwisdom@gmail.com	

Statement of integrity: By typing the names of all group members in the text boxes below, you confirm that the assignment submitted is original work produced by the group (excluding any non-contributing members identified with an "X" above).	
Team member 1	**Shailza Virmani**
Team member 2	**Harshil Sumra**
Team member 3	**Wisdom Yakano**

Use the box below to explain any attempts to reach out to a non-contributing member. Type (N/A) if all members contributed. **Note:** You may be required to provide proof of your outreach to non-contributing members upon request.
N/A

Individual answers to Steps1 and 2 - Introduction of Data Structures

Definition -

It's an organized process to store and retrieve data and helps to write complex programs and it can be accessed more efficiently. There are different types of data structures for different purposes.

Types of data structures :

Basic types of data structures include - Lists, Sets, Dictionaries, Tuple. Each of them is unique in their own way.

Lists : These are just like arrays used in other languages. It's flexible in nature as the items in the lists need not be of the same type. These are used for storing multiple items in a single variable. We use square brackets to store the data separated by comma. These are dynamic, mutable, ordered and lists allow duplicates.

Lists can be declared as -

```
list_1 = [1, 2, 3, 4, 5]
```

We can also use the 'for' loop inside a list.

Let's consider a sample example for list using a for loop

```
INPUT-
elements = [ele for ele  in range(10)]
print(elements)
OUTPUT-
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Will store all the elements of the range 0 to 9

Elements of the lists are accessed by index.
Such as : list_1[0] will print 1 to the console which is the first element of the list_1
These are also known as arrays or sequences. Items of any data type can be added or removed from the list. We can also join two lists together by just using + operator as we perform operations on numbers or strings

Methods used in lists :

append() : this method will append one element at the end of the list
copy() : returns a copy of lists
sort(): sorts the list alphanumerically
extend(): add elements of the list to the end of the current list
count(): returns the specified number of times a value occurs in the list.
insert(): adds an element at a specified position
reverse(): reverses the order of the list

Dictionaries : It contains key-value pair data. It is an ordered collection of data and doesn't allow duplicates. It is created by placing a sequence of elements within curly **{}** braces, separated by 'comma'. Dictionary keys are case sensitive, if they have the same name but different cases of Key will be treated distinctly.

Example -
INPUT :

```
dict_1 = {'name': 's', 'class': 'wqufe'}
print(dict_1)
```

OUTPUT :

```
{'name': 's', 'class': 'wqufe'}
```

Nested dictionary :

It stores a dictionary inside a dictionary.
E.g code:

```
dict_2 = {'name': 'shailza', 'class': {'finance':'wqufe', 'programming': 'ds_module'}}

print(dict_2)
```

OUTPUT

```
{'name': 'shailza', 'class': {'finance': 'wqufe', 'programming': 'ds_module'}}
#Here we are printing the dictionary as it is.
```

print(dict_2['class'])
# Here we are trying to print the second element of the dictionary that we declared.
OUTPUT
{'finance': 'wqufe', 'programming': 'ds_module'}

print(dict_2['class']['finance'])
In this example, we can see how the second nested key value is used to print the element of a dictionary
OUTPUT
wqufe

Dictionary Methods:

There is a set of built in methods that we can use in a dictionary-
clear()- This will remove all elements of the dictionary
pop()- Removes element with the specified key
copy()- Returns a copy of a dictioanry
update()- It updates the dictionary with a stated number of key-value pairs.

Similarly there are many more methods that we can use to work with dictionaries.

Tuple : It's used to store multiple items in a single variable. It's ordered and can not be updated or changed. It allows duplicates of data. It is similar to a list in terms of indexing, nested objects, and repetition but a tuple is immutable while lists are mutable.For tuples we use round brackets.

Example -

INPUT:
tuple_1 = (3,4,5,6,7)
print(tuple_1)
OUTPUT:
(3, 4, 5, 6, 7)

If we need to add some items in the tuple then it can not be done directly as we do in lists but we have to first convert it to a list and then we can use the append function to do the same.
Example:

```
INPUT:
 tup_2 = list(tuple_1)
 print(tup_2)
OUTPUT:
[3, 4, 5, 6, 7]
INPUT:
tup_2.append(10)
 print(tup_2)
OUTPUT:
[3, 4, 5, 6, 7, 10]
INPUT:
tuple_2 = tuple(tup_2)
print(tuple_2)
OUTPUT:
(3, 4, 5, 6, 7, 10)
```

As we can see in this question the tuple is updated by first converting that to list and then to tuple back.

We can also iterate items in the tuple using a for loop

```
Example:
INPUT:
for x in tuple_2:
print(x)
OUTPUT:
3
4
5
6
7
10
```

Tuples can also be joined together using ' + '

INPUT:
tuple_2 + tuple_2
OUTPUT:
(3, 4, 5, 6, 7, 10, 3, 4, 5, 6, 7, 10)

Tuples can also be multiplied n times

INPUT:
tuple_2 * 5
OUTPUT:
(3, 4, 5, 6, 7, 10, 3, 4, 5, 6, 7, 10, 3, 4, 5, 6, 7, 10, 3, 4, 5, 6, 7, 10, 3, 4, 5, 6, 7, 10)

Tuple methods :

In tuple we have only two built-in methods that we can use -
count() - It counts the number of times a value has occurred.
index() - returns the index or position of the element

Applications in the real world:

One of the most elegent application of list and dictionary data structures is when we store data, both structured or unstructured data in 'JSON' format. JSON format is nothing but a nested combination of lists and dictionaries. Dictionaries can also be used in encription/decryption of information by utilizing dictionary's highly accessible key-value structure for bot encoding and decoding, atleast for the simple cases of encription, like substitution : in which we just word replace each character/word with a randomly decided number, word or even a symbol. A very nice example of utilization of List's flexibility is Deque dataq structure. It basically adds the functionality of append and pop at both ends of a list. The we can easily utilize this deque as both an example of queue or stacks. A queue is basically a list that follows the principle of 'first in first out' while stack follows ' Last in first out'. So basically, all kinds of undo applications utilize this stacks applications. Therefore, when we press undo out last recorded action gets reverted. An application queue is in the online retail market which would ensure that most recently placed order gets fullfilled first.

.

Advantages and Disadvantages :

Dictionaries:

Advantages	Disadvantages
It improves the readability of the code.	These are unordered.
To quickly access the value, it links the value to the unique key.	These take more space than other data structures.

Lists:

Advantages	Disadvantages
To store multiple items in a single variable.	These can be appended only at the end.
Ordered, Mutable, Dynamic,	It is not a linked list at all.

Tuples:

Advantages	Disadvantages
These are faster than lists	Cannot add, sort, remove, replace an element in tuple.
Make the code safe from accidental modification	These are immutable and so cannot be changed.

Study-aid / crib-sheet for best practices :

List/Tuple :

Creating an empty list /tuple:
Example:
Lists: cribList = []

Tuples: cribTuple = ()

Update an item of List/tuple:
Example:
Lists: cribList[x] = "x“
Tuples: tuples are immutable!

Access an element in position x in list/tuple:
Example:
"x" in cribListOrTuple

Remove element in position X of list/tuple:
Example:
Lists: del cribList[x]
Tuples: tuples are immutable!

Change a list/tuple to tuple/list:
Example:
List to Tuple: tuple(cribList)
Tuple to List: list(cribTuple)

Dictionaries :

Creating an empty Dict
Example:
cribDict = {}

Adding an element with key "x" to the Dict
Example:
cribDict["x"] = value

Modify the element with key "x"
Example:
cribDict["x"] = newValue

Validate if the dictionary has key "x"
Example:
"x" in cribDict

Obtain the list of keys
Example:
cribDict.keys()

Check the size of the dictionary
Example:
len(cribDict)

Remove an element with key "x" from the dictionary
Example:
del cribDict["k"]

Remove all the elements in the dictionary
Example:
cribDict.clear()

Reference
https://www.geeksforgeeks.org/python-data-structures/