Week- 2: Python Data Structures

Introduction:-

In this session, we will extend our discussion to the essential data structures frequently used in data analysis. we will go into the depths of data structures- basically, structures that can hold the data together.

A data structure is a particular way of organising data in a computer so that it can be used effectively. We will cover the four builtin data structures in Python i.e., **tuples**, **lists**, **dictionaries**, and **sets**, in a fair amount of depth.

Lists:-

In this session, we will understand more about data structure; A data structure is nothing but a collection of **data values**.

Practically, whenever you are handling data, you are not given a single number or a string; instead, you are given a set of values and asked to manage them. There are multiple data structures that can handle such a set of values, each with its unique properties.

Lists are the most basic data structure available in python that can hold multiple variables/objects together for ease of use.
There are different ways of accessing elements that are present in a list. You can use indexing to access individual elements from a list or you can use slicing to multiple elements. Implement it yourself using the code given below:

```
# Indexing example
L = ["Chemistry", "Biology", [1989, 2004], ("Oreily", "Pearson")]
L[0]

# Slicing
L[0:3]
```

For more examples of indexing and slicing in lists, you can refer to the python documentation on lists.

You can check the members of a list by using the "in" keyword. The membership check operation returns a boolean output. Lists are **mutable**, which means the elements of a list can be changed.

Some of the essential methods available with lists include:

- **extend():** Extend a list by adding elements at the end of the list.
- **append():** Append an object to the end of a list.

The significant difference between the two methods is that the

append() method takes an object passed as a single element and adds it to the end of the list, whereas extend() takes an object passed as an iterable and adds every element in the iterable at the end of the list. Take a look at the code below and implement it yourself:

```
# extend()
L = ["Chemistry", "Biology", [1989,  2004] ,("Oreily" , "Pearson")]
L.extend([5, 8])
L

# append()
L = ["Chemistry", "Biology", [1989,  2004], ("Oreily" , "Pearson")]
L.append([5, 8])
L
```

In the examples above, you can see that when a list is passed in the extend method, it takes each element from the list and appends it to the end of the list. However, when the same list is passed to an append method, it considers this list as a single element and appends it to the end of the list.

Some of the common list functions that you have learnt in the video above:

- **pop(index)**: Remove and return the item at index (default last)
- **remove(value)**: Remove the first occurrence of a value in the list

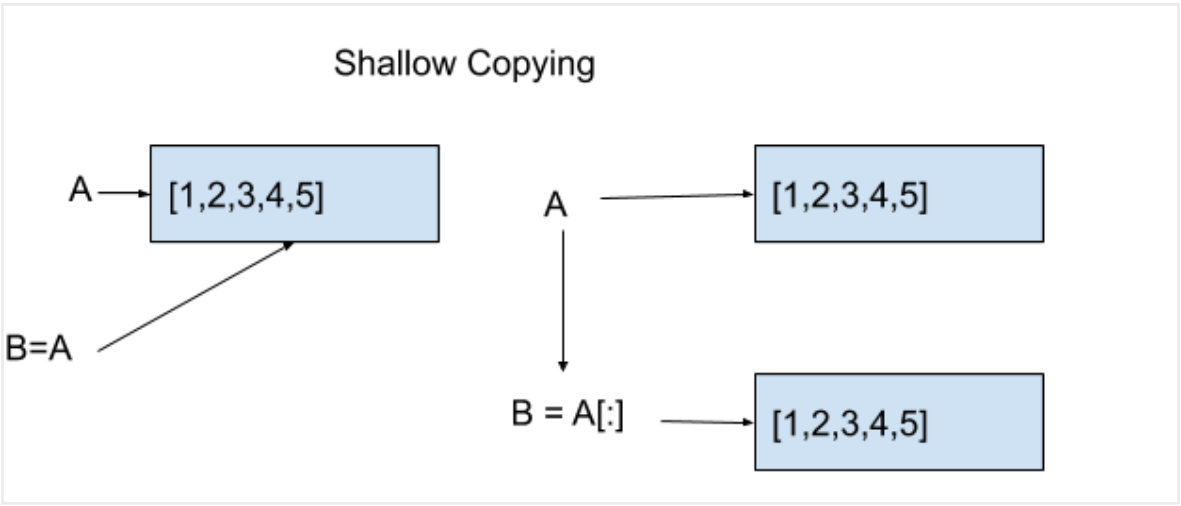
Till now you have learnt indexing, slicing, adding elements, and removing elements from a list.

There are two functions you can use to sort a list:

- **sort():** Sorts the elements of a list in place
- **sorted():** Assign the sorted elements to a new list and the original list is left as is.

Shallow/Shadow Copy in Python:-

Finally, we have to learn about shallow copying, which is an important concept to understand while handling lists. As you saw, by assigning B = A, you refer to the same list object in the memory, and the changes made to list A will be reflected in list B as well. With A[:], you are creating a new object, and this new object is assigned to B; here, any changes in list A would not affect list B anymore. Take a look at the image below to understand better:



Tuples:-

Tuples are similar to lists in almost of their functionalities, but there is one significant difference in both the data structures lists are mutable and tuples are not. Let's start learning about tuples.

A **tuple** contains a sequence of comma-separated values within parentheses. An important feature of a tuple is immutability, which means the elements of a tuple **cannot** be altered. It is used to store data such as employee information, which is not allowed to be changed.

For example: ('Gupta', 24 , 'Project Manager')
Here, we are storing the employee information name, age and designation in the form of a tuple. This makes this information unchangeable or immutable.

One crucial point to observe from the list of examples covered in the video is that a tuple can also be defined without using parentheses. For example:

X = 1, 2, 3, 4 → makes X a tuple

So basically what I'm saying is, there are 3 ways to define a tuple:-

```
# multi valued tuple with parantheses
t = (1,2,3,4)
print(type(t))

# multi valued tuple without parentheses

t = 1,2,3,4
print(type(t)

# single valued tuple
```

t = 1,
print(type(t)) O/P - tuple

Even if you try to use parentheses for creating a single valued tuple you won't be able to do so.

single valued tuple I.e. without parentheses

t = (1)
print(type(t)) O/P - int

Accessing the elements of tuples is similar to accessing elements in a list. You can use the same indexing method. In indexing of list, each character is assigned an index; similarly, each element is given an **index** in a tuple.

Tuples are ordered sequences, which means the order in which the elements are inserted remains the same. This makes them flexible for indexing and slicing just like lists. Using slicing, you were able to obtain sections of the data from a list; here, you will be able to obtain a subset of elements.

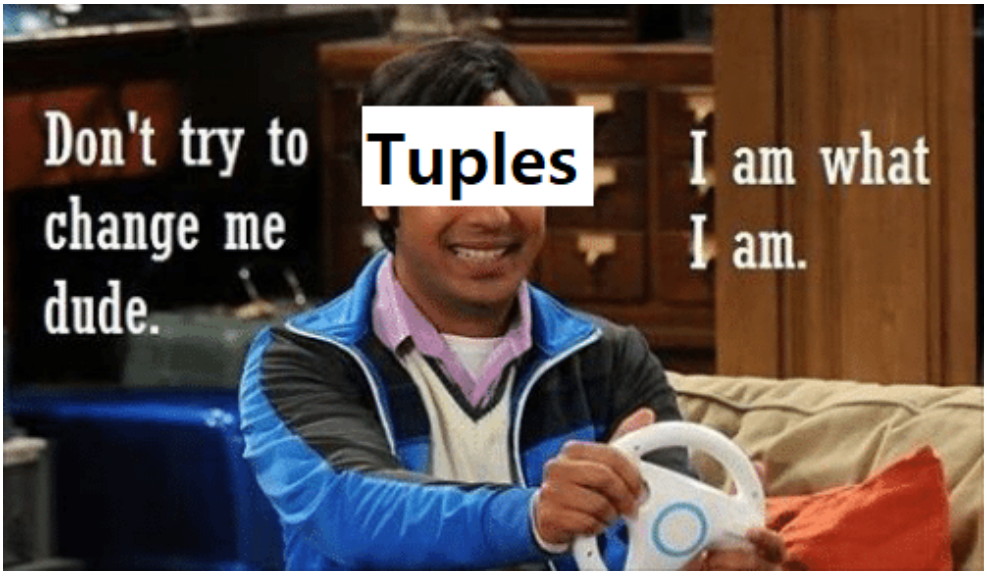
Immutability is the differentiating property between lists and tuples. The elements in a tuple cannot be changed once the tuple is declared. You have to make a new tuple using concatenation if you wish to change a value in a given tuple. You also saw tuples can be sorted just like lists.

you learnt that a tuple can have an iterable object or another sequence as its elements.

t = (**1**,**5**,"Disco", ("Python", "Java"))

If you apply the type function on the third element, it would return a tuple:
t = (**1**,**5**,"Disco", ("Python", "Java"))
type(t[**3**])
tuple

Towards the end of the video, you saw how the inbuilt **dir()** function helps to look for the list of methods that can be used while handling tuples. The **dir()** function only gives the list of methods available; instead, using the **help()** function and passing an empty tuple gives you a brief description of each of the methods available with tuples:



Tuples are immutable. Don't try to change them!

So far you should have a good understanding of both tuples and lists. In the next segment, you will be introduced to sets in python.

Sets:-

In the earlier segments, you learnt about lists and tuples, which are ordered sequences. In this segment, you will learn about a data structure '**sets**' that is unordered, or, in other words, it doesn't maintain an order in which elements are inserted. This makes sets unfit for indexing or slicing, but what is their use then?

By now, you would have found the answer to our earlier question about why sets are used: They can eliminate duplicates. This feature is necessary while handling massive data where most of it is just redundant and repeating. Let's take a look at an example to understand this better:

Let's say you have a huge list that contains student grades:

```
Grades = ['A', 'A', 'B', 'C', 'D', 'B', 'B', 'C', 'D', 'E', 'C', 'C', 'A', 'B', 'F', 'D',
'C',
        'B', 'C', 'A', 'B', 'F', 'B', 'A', 'E', 'B', 'B', 'C', 'D'...]
```

You want to identify distinct grades allotted to students. Obviously, you cannot check every element of this list; instead, we make use of sets which gets our job done here.

By using the set function on the grades, you would get the distinct

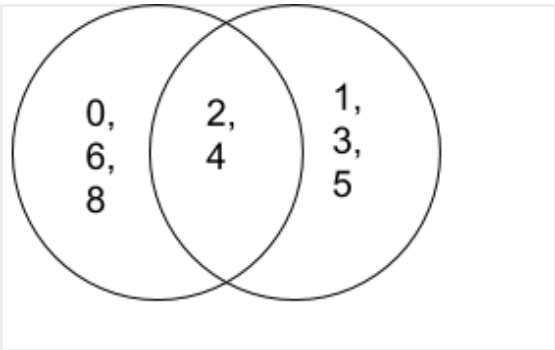
grades in the form of a set:
Grades = ["A", "A", "B", "C", "D", "B", "B", "C", "D", "E", "C", "C", "A", "B", "F", "D", "C", "B", "C", "A", "B", "F", "B", "A", "E", "B", "B", "C", "D"]

set(Grades)
{'A', 'B', 'C', 'D', 'E', 'F'}

With all the conceptual understanding of the application of sets let's learn to declare sets and add and remove elements from them

These sets further help you perform all the typical set operations that you learnt in high school.

Imagine you have two sets:
A = {0,2,4,6,8}
B = {1,2,3,4,5}



Set methods

- Union represents the total unique elements in both sets.
  - **A.union(B)** → {0, 1, 2, 3, 4, 5, 6, 8}
- Intersection represents the elements common to both sets.
  - **A.intersection(B)** → {2, 4}
- Difference(A-B) represents the elements present in A and not in B.
  - **A.difference(B)** → {0, 6, 8}
- Symmetric difference represents the union of the elements A and B minus the intersection of A and B.
  - **A^B →** {0, 6, 8, 1, 3, 5}

The order of elements shown above may not be the same when you actually execute the above operations. It is because sets are unordered, which means they do not store the positional index of an element in the set.

A Fun Activity

Try to decode the set operation give below:
(A.union(B)).difference(A.intersection(B))

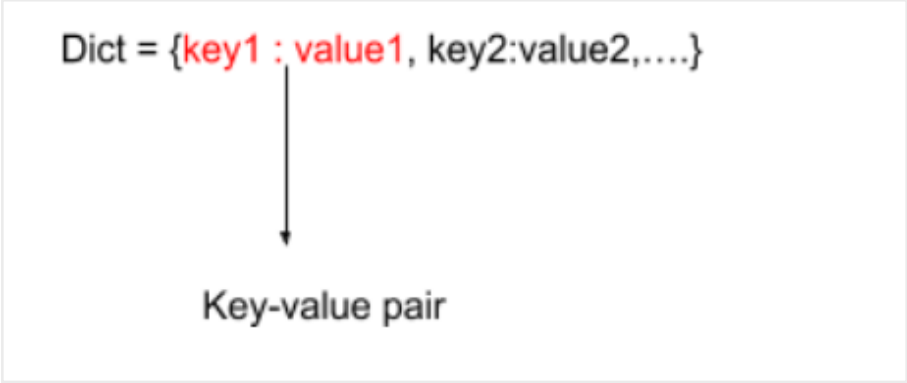
Instead of using commands, you can also use simple mathematical operators between sets to perform the various operations. Go through the below link to understand how:
Operations on Sets

Dictionaries:-

The first thing that comes to mind when you hear about dictionaries is the Oxford Dictionary where you can look up the meanings of words. So, you can imagine how dictionaries work. A dictionary is a collection of words along with their definitions or explanations.

At a broader level, you could describe a dictionary as a mapping of words with their synonyms or meanings.

The dictionary structure looks as shown below:



Dictionary is one data structure that is very useful because of the way in which it is defined. Let's take a look at a small example to understand this. Imagine we have employee data with the following attributes: employee id, name, age, designation. Now, let's say you want to retrieve employee details based on employee id (since this is unique to each employee). How would you store the data?

E_id	Employee Information
101	Akash, 24, Content Strategist
102	Vishal, 30, Project Manager
.....	.....
159	Vikas, 18, Intern

Let's say you use a list or tuple here; it would simply be difficult. Let's see how.

First, you should have a list where each element represents an employee, and this element should be iterable again since you have different values to represent every employee.

[[e_id1, name1, age1, designation1],[e_id2, name2, age2, designation2]...]

But would this serve the purpose? There is a problem here: **e_id** is unique and cannot be changed, or, in other words, it is an immutable entity. Let's say we make this entire element a tuple:

[(e_id1, name1, age1, designation1),(e_id2,name2, age2, designation2)...]

Now, this would make **name**, **age** and **designation** immutable entities; instead, they should be mutable entities:

[(e_id1, [name1, age1, designation1]),(e_id2,[name2, age2, designation2])...]

Having arrived at this point, imagine how it is to retrieve information about a particular employee based on their employee id. To achieve this, you need to use the loops concepts in Python, but isn't this whole thing difficult? Imagine how simple this would be if you used a dictionary here:

E = { e_id1 : [name1, age1, designation1],e_id2 : [name2, age2, designation2],...}

Here -

- **e_id** is unique;
- **name**, **age** and **designation** are mutable; and
- simply using **E[e_id1]** will give employee e_id1's information.

Now that you understand the application and use of a dictionary, let's learn to declare a dictionary and also explore some of its features:

To conclude, in this session we first had a look at various data structures supported by Python which are tuples, list, sets and dictionaries and then we understood the use-case or say features each one of these data structures offer. Hope this was an insightful session wherein we saw certain real-world application based use-cases and various operations supported by these data structures.

In the next session, we will dive deep into control structures and functions where you will learn about programming the decision-making capabilities and automating tasks via several *constructs* and *loops* supported by python.