What are the characteristics of the tuples? Is tuple immutable?

The characteristics of tuples are:
1. A tuple can store any data types ,for example string, integer, float, list etc .
2. A tuple is identified in first braces.
3. Tuples are immutable, thus you cannot reassign a value inside a tuple.
4. Tuples in python has only two inbuilt method that is count() and index()

Yes, tuples are immutable datatypes. Thus, data reassignment is not possible in tuples.

What are the two tuple methods in python? Give an example of each method. Give a reason why tuples have only two in-built methods as compared to Lists.

The two tuples methods which are present in python are count() and index().
An example of count() is:-
a=(1,2,3,1,2,4,5,6,6,7,8,"angs",'human').
print(a.count(1))
print(a.index(1))
the output will be the count of the given arguement in the tuple which is 2
whereas the index method will return the first occurance of the given arguement in the tuple that is 0

Tuples and lists are both used for storing collections of items in Python, but they have some fundamental differences. One of the key differences between tuples and lists is that tuples are immutable, meaning that their contents cannot be modified after creation, whereas lists are mutable and can be modified.
Because tuples are immutable, they only need a limited set of built-in methods that are relevant to their usage. The two built-in methods of tuples are count() and index().
The count() method is used to count the number of occurrences of a specified item in a tuple, while the index() method is used to find the index of the first occurrence of a specified item in a tuple.
On the other hand, lists have many more built-in methods compared to tuples, since they can be modified and manipulated more extensively. Some of the additional built-in methods of lists include append(), insert(), remove(), and sort(), among others.
In summary, tuples only have two in-built methods because they are immutable and do not require a lot of built-in methods for manipulation, unlike lists, which are mutable and require a broader range of built-in methods to be useful.

Which collection datatypes in python do not allow duplicate items? Write a code using a set to remove duplicates from the given list.
List = [1, 1, 1, 2, 1, 3, 1, 4, 2, 1, 2, 2, 2, 3, 2, 4, 3, 1, 3, 2, 3, 3, 3, 4, 4, 1, 4, 2, 4, 3, 4, 4]

set datatype is a collection datatype in python which does not allow duplicate items.

```
print(set(List))
```

# Explain the difference between the union() and update() methods for a set. Give an example of each method.

Both union() and update() methods are used for combining sets in Python, but they differ in how they modify the original set.
The union() method returns a new set that contains all the elements from the original set as well as the elements from one or more other sets.
The original set is not modified by the union() method.

Here's an example of using the union() method:

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
set3 = {5, 6, 7}

new_set = set1.union(set2, set3)

print(new_set)   # Output: {1, 2, 3, 4, 5, 6, 7}
print(set1)      # Output: {1, 2, 3} (original set is not modified)
```

On the other hand, the update() method modifies the original set by adding all the elements from one or more other sets to it.
Here's an example of using the update() method:

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
set3 = {5, 6, 7}

set1.update(set2, set3)

print(set1)   # Output: {1, 2, 3, 4, 5, 6, 7} (original set is modified)
```

In this example, set1 is modified by adding all the elements from set2 and set3 to it using the update() method.
In summary, the union() method returns a new set without modifying the original set, while the update() method modifies the original set by adding elements from other sets.

What is a dictionary? Give an example. Also, state whether a dictionary is ordered or unordered.

A dictionary is a built-in data type in Python that allows you to store a collection of key-value pairs.
Each key in a dictionary maps to a corresponding value, and you can use the key to look up the value associated with it.
In Python, dictionaries are created using curly braces {} and colons : to separate the keys and values. Here's an example of a dictionary:

```
my_dict = {'name': 'John', 'age': 30, 'location': 'New York'}

print(my_dict['name'])      # Output: John
print(my_dict['age'])       # Output: 30
print(my_dict['location'])  # Output: New York
```

In this example, my_dict is a dictionary that stores information about a person's name, age, and location.
The keys in the dictionary are strings ('name', 'age', and 'location'), and the corresponding values can be of any data type ('John' is a string, 30 is an integer, and 'New York' is a string).
Dictionaries are unordered in Python, which means that the items in the dictionary are not stored in any particular order.
When you iterate over a dictionary, the order in which the key-value pairs are returned may vary each time you run the code.
If you need to preserve the order of the items, you can use a collections.OrderedDict instead.

Can we create a nested dictionary? If so, please give an example by creating a simple one-level nested dictionary.

Yes, we can create a nested dictionary in Python. A nested dictionary is a dictionary that contains one or more dictionaries as values.

Each key in the outer dictionary maps to an inner dictionary, which can then have its own keys and values.

Here's an example of a simple one-level nested dictionary:

```
employee_data = {
   'John': {'age': 30, 'salary': 50000},
   'Jane': {'age': 25, 'salary': 45000},
   'Bob': {'age': 40, 'salary': 60000}
}

print(employee_data['John'])        # Output: {'age': 30, 'salary': 50000}
print(employee_data['Jane']['age']) # Output: 25
print(employee_data['Bob']['salary'])# Output: 60000
```

In this example, employee_data is a nested dictionary that contains information about three employees. Each employee is represented by an inner dictionary with keys 'age' and 'salary'. The outer dictionary maps employee names to their respective inner dictionaries.
To access the inner dictionary for a particular employee, you can use the key of the employee's name to look up the corresponding value in the outer dictionary.
Once you have the inner dictionary, you can use its keys to access the values of the 'age' and 'salary' keys.

Using setdefault() method, create key named topics in the given dictionary and also add the value of the key as this list ['Python', 'Machine Learning’, 'Deep Learning']

Here's an example of using the setdefault() method to add a new key named 'topics' to a dictionary and set its value to the list ['Python', 'Machine Learning', 'Deep Learning']:

```
my_dict = {'name': 'John', 'age': 30}

topics_list = ['Python', 'Machine Learning', 'Deep Learning']

my_dict.setdefault('topics', topics_list)

print(my_dict)
```

In this example, we first define a list called topics_list that contains the values we want to assign to the new 'topics' key.
We then use the setdefault() method to add the new key and set its value to the topics_list.
The output of this code will be:

```
{'name': 'John', 'age': 30, 'topics': ['Python', 'Machine Learning', 'Deep Learning']}
```

The setdefault() method checks if the key already exists in the dictionary. If it does, the method returns the value associated with the key. If the key does not exist, the method adds the key to the dictionary with the specified default value and returns the default value.
In this example, since the 'topics' key did not already exist in the dictionary, the method added it with the value of topics_list.

What are the three view objects in dictionaries? Use the three in-built methods in python to display these three view objects for the given dictionary. dict1 = {'Sport': 'Cricket' , 'Teams': ['India', 'Australia', 'England', 'South Africa', 'Sri Lanka', 'New Zealand']}

In Python, dictionaries have three view objects that provide a dynamic view of the dictionary's contents. These view objects are:

dict_keys: This view object contains the keys of the dictionary.
dict_values: This view object contains the values of the dictionary.
dict_items: This view object contains the key-value pairs of the dictionary as tuples.
Here's an example of how to use the three in-built methods in Python to display these view objects for the given dictionary:

```
dict1 = {'Sport': 'Cricket', 'Teams': ['India', 'Australia', 'England', 'South Africa', 'Sri Lanka', 'New Zealand']}

# Get the dict_keys view object
keys_view = dict1.keys()
print(keys_view)

# Get the dict_values view object
values_view = dict1.values()
print(values_view)

# Get the dict_items view object
items_view = dict1.items()
print(items_view)
```

## The output of this code will be:

```
dict_keys(['Sport', 'Teams'])
dict_values(['Cricket', ['India', 'Australia', 'England', 'South Africa', 'Sri Lanka', 'New Zealand']])
dict_items([('Sport', 'Cricket'), ('Teams', ['India', 'Australia', 'England', 'South Africa', 'Sri Lanka',
'New Zealand'])])
```

In this example, we define a dictionary called dict1 that contains information about a sport and its teams.
We then use the keys(), values(), and items() methods to get the view objects for the keys, values, and key-value pairs of the dictionary, respectively.
Finally, we print out the view objects to display their contents.