Everything in Python is an object. Each object has its own data attributes and methods associated with it. In order to use an object efficiently and appropriately, we should know how to interact with them.

Lists, tuples, and sets are 3 important types of objects. What they have in common is that they are used as data structures. In order to create robust and well-performing products, one must know the data structures of a programming language very well.

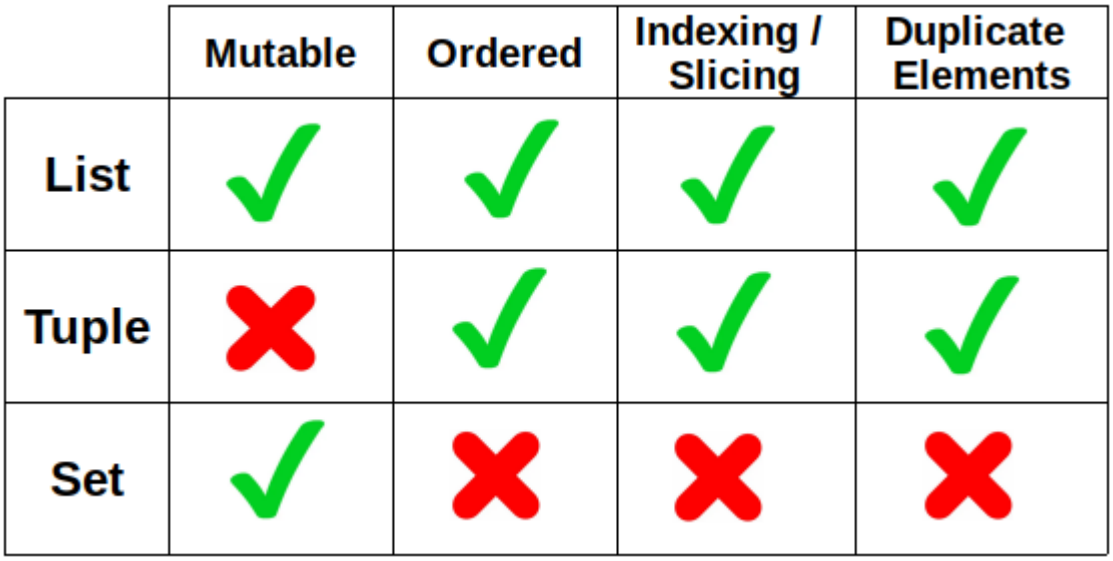
In this post, we will see how these structures collect and store data as well as the operations we can do with them. We will see both the differences and similarities between them.

Let's start by briefly explain what these objects are. We will then do the examples to go in detail for each one.

List is a built-in data structure in Python. It is represented as a collection of data points in square brackets. Lists can be used to store any data type or a mixture of different data types. Lists are mutable which is one of the reasons why they are so commonly used.

Tuple is a collection of values separated by comma and enclosed in parenthesis. Unlike lists, tuples are immutable. The immutability can be considered as the identifying feature of tuples.

Set is an unordered collection of distinct immutable objects. A set contains unique elements. Although sets are mutable, the elements of sets must be immutable. There is no order associated with the elements of a set. Thus, it does not support indexing or slicing like we do with lists.



List vs Tuple vs Set (image by author)

We now have a basic understanding of what these objects are. The following examples will cover how we can interact with these objects.

1. List vs Set

We can create a list or set based on the characters in a string. The functions to use are the list and set functions.

```
text = "Hello World!"print(list(text))
['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd',
'!']print(set(text))
{'H', 'W', 'o', ' ', 'l', 'r', '!', 'e', 'd'}
```

The differences in the resulting list and set objects:

The list contains all the characters whereas the set only contains unique characters.

The list is ordered based on the order of the characters in the string. There is no order associated with the items in the set.

2. List vs Set — indexing

In the previous example, we saw that sets do not possess an order. Thus, we cannot do slicing or indexing on sets like we do with lists.

```
text = "Hello World!"list_a = list(text)
print(list_a[:2])
['H','e']set_a = set(text)
print(set_a[:2])
TypeError: 'set' object is not subscriptable
```

Slicing or indexing on sets raise a TypeError because it is an issue related to a property of set object type.

3. List vs Tuple

The difference between list and tuple is the mutability. Unlike lists, tuples are immutable. For instance, we can add items to a list but cannot do it with tuples.

```
list_a = [1,2,3,4]
list_a.append(5)
print(list_a)
[1,2,3,4,5]tuple_a = (1,2,3,4)
tuple_a.append(5)
AttributeError: 'tuple' object has no attribute 'append'
```

The functions that changes a collection (e.g. append, remove, extend, pop) are not applicable to tuples.

4. Tuple — mutable elements

The immutability might be the most identifying feature of tuples. We cannot assign to the individual items of a tuple.

```
tuple_a = (3, 5, 'x', 5)
tuple_a[0] = 7 #error
```

Although tuples are immutable, they can contain mutable elements such as lists or sets.

```
tuple_a = ([1,3], 'a', 'b', 8)
tuple_a[0][0] = 99
```

```
print(tuple_a)
([99, 3], 'a', 'b', 8)
```

5. Del function

Del function stands for delete so it is used to delete an item from a collection. It takes the index of the item to be deleted.

Since sets are unordered, they do not have an index of items in them. Thus, del function cannot be used on a set.

```
list_a = [1, 2, 3, 4]
del(list_a[0])print(list_a)
[2, 3, 4]
```

Note: There are two ways to index a list:

From the start to end: 0, 1, 2, 3

From the end to start: -1, -2, -3

6. Remove function

Unlike the del function, the remove function can be used on both lists and sets. We pass the item to be removed rather than its index.

```
list_a = ['a','b',3,6]
list_a.remove('a')
print(list_a)
['b', 3, 6]set_a = {'a','b',3,6}
set_a.remove('a')
print(set_a)
{3, 6, 'b'}
```

7. Discard function

Discard can also be used to remove an item from a set. Lists do not have discard attribute though.

The difference between “remove” and “discard” is observed when we try to remove an item that is not in the set. Remove will raise an error but nothing happens with discard.

```
#remove
a = {1,2,3}
a.remove(5)
KeyError: 5
#discarda = {1,2,3}
a.discard(5)
print(a)
{1,2,3}
```

8. Pop function

Pop function can be used on both lists and sets. However, it works differently on lists and sets.

By default, the pop function removes the last item from a list and returns it. Thus, we can assign it to a variable. We can pass an index to the pop function remove an element at a specific index. For instance, pop(-2) will remove the second item from the end.

```
list_a = ['a','b',3,6,4]
item = list_a.pop()print(list_a)
['a', 'b', 3, 6]print(item)
4
```

When used on a set, the pop function removes an arbitrary item since there is no index or order in a set.

```
set_a =  {'a','b',3, 6, 4}
item2 = set_a.pop()print(set_a)
{4, 6, 'a', 'b'}print(item2)
3
```

9. List or Set or Tuple

These collection objects can be converted from one to another. The functions to use are, as the names suggest, list, tuple, and set.

```
a = [1,2,3,'a',1,3,5]print(tuple(a))
(1, 2, 3, 'a', 1, 3, 5)print(set(a))
{1, 2, 3, 5, 'a'}b = {'a',1, 4, 8}print(list(b))
[8, 1, 4, 'a']print(tuple(b))
(8, 1, 4, 'a')
```

10. Adding new items

Since tuples are immutable, we can only add new items to a list or set. We have more alternatives when altering a list due to being ordered or indexed.

For instance, append method adds an item at the end of a list. Since sets do not possess the concept of end or start, we cannot use the append method. For sets, the add method is used to add new items.

```
a = [1,2,3]
a.append(4)
print(a)
[1,2,3,4]b = {1,2,3}
b.add(4)
print(b)
{1,2,3,4}
```

11. Insert an item to a list

The insert function is also used to add an element to a list. However, it allows specifying the index of the new element. For instance, we can add a new element at the beginning of the list (index=0).

```
a = [1, 2, 3, 4, 5]
a.insert(0, 'a')
a
['a', 1, 2, 3, 4, 5]
```

Since it requires an index, we cannot use the insert function on sets.

12. Combining two objects

There will be cases where we need add items of a type together. We have multiple options to combine objects of lists, tuples, and sets.

The "+" operator can be used to add lists or tuples but not sets.

```
a = [1,2,3]
b = [11,32,1]print(a + b)
[1, 2, 3, 11, 32, 1]print(tuple(a) + tuple(b))
(1, 2, 3, 11, 32, 1)
```

We can use the union operator to combine two sets. The duplicate elements will be removed.

```
a = {1,2,3,4}
b = {1,5,6}print(a.union(b))
{1, 2, 3, 4, 5, 6}
```

Note: The set notation is similar to the dictionary notation in Python. The difference is that when creating dictionaries, we put key-value pairs inside curly braces instead of single items.

We need to keep that in mind when creating an empty dictionary. If we only use curly braces with nothing inside, Python thinks it is an empty dictionary. We can use the set function to create an empty set.

```
a = {}print(type(a))
<class 'dict'>b = set()
print(type(b))
<class 'set'>c = set({})
print(type(c))
<class 'set'>
```

13. Sorting

We can talk about sorting only if there is an order. Thus, sorting applies to lists and tuples. Sets cannot be sorted since there is no order.

The sort function modifies the object it is applied. Thus, we can only use it on lists. Tuples are immutable so we cannot sort them.

```
a = [3,1,5,2]
a.sort()
print(a)
[1, 2, 3, 5]
```

However, we can use the sorted function on tuples. It creates a sorted list of any iterable. So we can use it to create a sorted list based on a tuple.

```
b = (6,1,4,2)
print(sorted(b))
[1,2,4,6]
```

sort(): sorts the object but does not return anything.

sorted(): returns a sorted list of the items in an iterable but does not modify the original object.

14. Updating a set

The update method can be used to update a set by the items in other iterables. Due to the nature of sets, the duplicate items are removed when updating.

```
a = {'x', 1, 4}
b = [3, 4, 1]
c = ('x', 'y', 'z')a.update(b,c)
print(a)
{1, 3, 4, 'y', 'z', 'x'}
```

15. Len and count

The len function returns the length (i.e. number of items) of a collection. It works on lists, tuples, and sets.

The count function can be used to count the number of occurrences of a specific element. It is only used with lists and tuples. Since sets do not contain any duplicate items, the count is 1 for all items.

```
a = [1,4,5,6,1]
b = (3,4)
c = {1,2,3,4}print(len(a), len(b), len(c))
5 2 4print(a.count(1), b.count(3))
2 1
```

Conclusion

We have covered the differences and similarities between 3 essential data structures in Python. I tried to design the examples to highlight the important points to keep in mind when interacting with these objects.

There are more methods and operations that can work on these objects. For instance, list comprehensions are quite useful in data analysis and manipulation.

A comprehensive understanding of data structures is highly important because they structures are fundamental pieces of any programming language. They are also key factors in designing algorithms.