Lecture 10

Data types (Tuple & sets)

Tuple

A tuple is a container which holds a series of comma-separated values (items or elements) between parentheses such as an (x, y) co-ordinate. Tuples are like lists, except they are immutable (i.e. you cannot change its content once created) and can hold mix data types. Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is ordered and unchangeable.

Tuples are written with round brackets.

Creating a Tuple

A tuple can be written as the collection of comma-separated (,) values enclosed with the small () brackets. The parentheses are optional but it is good practice to use. A tuple can be defined as follows. An empty tuple can be created.

Note: Tuple items can be of any data type, i.e. string, int and boolean data types. A single tuple can contain different data types.

```
>>> T1 = (101, "Peter", 22)
>>> T2 = ("Apple", "Banana", "Orange")
>>> T3 = 10,20,30,40,50
>>> print(type(T1))
<class 'tuple'>
>>> print(type(T2))
<class 'tuple'>
>>> print(type(T3))
<class 'tuple'>
>>> T4=()
```

Creating a tuple with single element is slightly different. We will need to put comma after the element to declare the tuple.

```
>>> T5=("karachi")
>>> print(type(T5))
<class 'str'>
>>> T5=("karachi",)
>>> print(type(T5))
<class 'tuple'>
>>>
```

Items and Index

Tuple items are ordered, unchangeable, and allow duplicate values.
Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Allow Duplicates

Since tuples are indexed, they can have items with the same value.

```
>>> T5=('karachi','karachi')
>>> print(type(T5))
<class 'tuple'>
>>>
```

Tuple Length

To determine how many items a tuple has, use the **len()** function:

```
>>> T5=('karachi','karachi')
>>> print(type(T5))
<class 'tuple'>
>>> print(len(T5))
2
```

tuple() constructor

It is also possible to use the tuple() constructor to make a tuple.
NOTE: the double round brackets.

```
>>> T=tuple(("apple", "banana", "cherry"))
>>> print(type(T1))
<class 'tuple'>
>>> T=tuple("apple", "banana", "cherry")
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    T=tuple("apple", "banana", "cherry")
TypeError: tuple expected at most 1 argument, got 3
>>>
```

Access Items

You can access tuple items by referring to the index number, inside square brackets:

```
>>> T
('apple', 'banana', 'cherry')
>>> print(T[0])
apple
>>> print(T[2])
cherry
>>>
```

Note: the first item has zero index.

Negative Indexing

We have already seen negative indexes in the previous lecture. The same rule apply in tuples too. Negative indexing means start from the end.

-1 refers to the last item, -2 refers to the second last item etc.

```
>>> T
('apple', 'banana', 'cherry')
>>> print(T[-2])
banana
>>> print(T[-1])
cherry
>>> print(T[-3])
apple
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range. When specifying a range, the return value will be a new tuple with the specified items. By leaving out the start value, the range will start at the first item. Similarly, by leaving out the end value, the range will go on to the end of the list.

```
>>> T3
(10, 20, 30, 40, 50)
>>> print(T3[2:4])
(30, 40)
>>> print(T3[1:4])
(20, 30, 40)
>>> print(T3[:4])
(10, 20, 30, 40)
>>> print(T3[:4])
(10, 20, 30, 40)
```

Note: The search will start at index 1 (included) and end at index 4 (not included).

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the tuple:

```
>>> T1=("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
>>> print(T1[-4:-1])
('orange', 'kiwi', 'melon')
>>>
```

This example returns the items from index -4 (included) to index -1 (excluded)

Item Existence

To determine if a specified item is present in a tuple use the 'in' keyword:

CODE

```
T1=("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
if "apple" in T1:
  print("Yes, it is in the tuple")
```

OUTPUT

```
>>>
==== RESTART: C:\Users\Her
Yes, it is in the tuple
>>>
```

Update Tuple

Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created. Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
>>> L=list(T1)
>>> L
['apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango']
>>> print (type(L))
<class 'list'>
>>> x=tuple(L)
>>> x
('apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango')
>>> print(type(x))
<class 'tuple'>
>>>
```

Add Items

Since tuples are immutable, they do not have a build-in append() method, but there are other ways to add items to a tuple.

1. Convert into a list:

Just like the workaround for changing a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

CODE

```
T1=("apple", "banana", "cherry", "kiwi", "melon", "mango")
y = list(T1)
y.append("orange")
T1 = tuple(y)
```

OUTPUT

```
>>> T1
('apple', 'banana', 'cherry', 'kiwi', 'melon', 'mango')
>>>
==== RESTART: C:\Users\Hera Noor\AppData\Local\Programs\Python\Pyth
>>> T1
('apple', 'banana', 'cherry', 'kiwi', 'melon', 'mango', 'orange')
>>> T1
```

2. Add tuple to a tuple:

You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

```
>>> T1 =('apple', 'banana', 'cherry', 'kiwi', 'melon', 'mango')
>>> T2=10,20,30,40,50
>>> T1+T2
('apple', 'banana', 'cherry', 'kiwi', 'melon', 'mango', 10, 20, 30, 40, 50)
>>>
```

Remove Items

Tuples are unchangeable, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items:

CODE

```
T1=("apple", "banana",'orange', "cherry", "kiwi", "melon", "mango")
y = list(T1)
y.remove("orange")
T1 = tuple(y)
print(T1)
```

OUTPUT

```
==== RESTART: C:\Users\Hera Noor\AppData\Local\Programs\Python\Python
('apple', 'banana', 'orange', 'cherry', 'kiwi', 'melon', 'mango')
('apple', 'banana', 'cherry', 'kiwi', 'melon', 'mango')
>>>
```

Remove Items

Tuples are unchangeable, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items.

DELETE TUPLE

Or you can delete the tuple completely. The 'del' keyword can delete the tuple completely.

```
>>> T1
('apple', 'banana', 'cherry', 'kiwi', 'melon', 'mango')
>>> y = list(T1)
>>> y.remove("apple")
>>> T1=tuple(y)
>>> T1
('banana', 'cherry', 'kiwi', 'melon', 'mango')
>>> del(T1)
>>> T1
Traceback (most recent call last):
  File "<pyshell#55>", line 1, in <module>
    T1
NameError: name 'T1' is not defined
>>>
```

Tuple Unpack

When we create a tuple, we normally assign values to it. This is called "packing" a tuple. But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking".

```
>>> fruits = ("apple", "banana", "cherry")
>>> (green, yellow, red) = fruits
>>> print(green)
apple
>>> print(yellow)
banana
>>> print(red)
cherry
```

Note: The number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.

Using Asterisk*

If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list.

```
>>> fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
>>> (green, yellow, *red) = fruits
>>> print(green)
apple
>>> print(yellow)
banana
>>> print(red)
['cherry', 'strawberry', 'raspberry']
```

If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

```
>>> fruits = ("apple", "mango", "papaya", "pineapple", "cherry")
>>> (green, *tropic, red) = fruits
>>> print(green)
apple
>>> print(tropic)
['mango', 'papaya', 'pineapple']
>>> print(red)
cherry
```

Loop in Tuple

You can loop through the tuple items by using a for loop.

CODE

```
T1 = ("apple", "banana", "cherry")
for x in T1:
  print(x)
```

OUTPUT

```
==== RESTART: C:\Users\Hera Noo
apple
banana
cherry
>>>
```

Loop Through the Index Numbers

You can also loop through the tuple items by referring to their index number.

Use the range() and len() functions to create a suitable iterable.

CODE

```
T1 = ("apple", "banana", "cherry")
for i in range(len(T1)):
  print(T1[i])
```

OUTPUT

```
==== RESTART: C:\Users\Hera
apple
banana
cherry
>>>
```

Join Tuple

Two tuples can be added, as well as multiplied. To join two or more tuples you can use the + operator. If you want to multiply the content of a tuple a given number of times, you can use the * operator. Lets have some examples.

```
>>> T1
('apple', 'banana', 'cherry')
>>> T2=10,20,30,40
>>> T=T1+T2
>>> T
('apple', 'banana', 'cherry', 10, 20, 30, 40)
```

```
>>> A=T2*3
>>> A
(10, 20, 30, 40, 10, 20, 30, 40, 10, 20, 30, 40)
>>> B=T1*2
>>> B
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
>>>
```

Tuple Methods

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Differences between Lists and Tuples

In most cases, lists and tuples are equivalent. However, there are some important differences to be explored in this article.

Syntax Differences

The syntax of a list differs from that of a tuple. Items of a tuple are enclosed by parentheses or curved brackets (), whereas items of a list are enclosed by square brackets []. We declared a variable named list_, which contains a certain number of integers ranging from 1 to 10. The list is enclosed in square brackets []. We also created a variable called tuple_, which holds a certain number of integers. The tuple is enclosed in curly brackets (). The type() method in Python returns the data type of the data structure or object passed to it.

Differences between Lists and Tuples

Mutable List vs. Immutable Tuple

An important difference between a list and a tuple is that lists are mutable, whereas tuples are immutable. What exactly does this imply? It means a list's items can be changed or modified, whereas a tuple's items cannot be changed or modified. We can't employ a list as a key of a dictionary because it is mutable. This is because a key of a Python dictionary is an immutable object. As a result, tuples can be used as keys to a dictionary if required.

```
>>> L= ["Python", "Lists", "Tuples", "Differences"]
>>> T = ("Python", "Lists", "Tuples", "Differences")
>>> L[3] = "Mutable"
>>> L
['Python', 'Lists', 'Tuples', 'Mutable']
>>> T[3] = "Immutable"
Traceback (most recent call last):
  File "<pyshell#87>", line 1, in <module>
    T[3] = "Immutable"
TypeError: 'tuple' object does not support item assignment
>>>
```

Differences between Lists and Tuples

Size Difference

Since tuples are immutable, Python allocates bigger chunks of memory with minimal overhead. Python, on the contrary, allots smaller memory chunks for lists. The tuple would therefore have less memory than the list. If we have a huge number of items, this makes tuples a little more memory-efficient than lists.

For example, consider creating a list and a tuple with the identical items and comparing their sizes:

```
>>> L
['Python', 'Lists', 'Tuples', 'Differences']
>>> T
('Python', 'Lists', 'Tuples', 'Differences')
>>> print("Size of tuple: ", T.__sizeof__())
Size of tuple:  56
>>> print("Size of list: ", L.__sizeof__())
Size of list:  72
>>>
```

Key Similarities

- They both hold collections of items and are heterogeneous data types, meaning they can contain multiple data types simultaneously.
- They're both ordered, which implies the items or objects are maintained in the same order as they were placed until changed manually.
- Because they're both sequential data structures, we can iterate through the objects they hold; hence, they are inerrable.
- An integer index, enclosed in square brackets [index], can be used to access objects of both data types.

Data Type – Sets

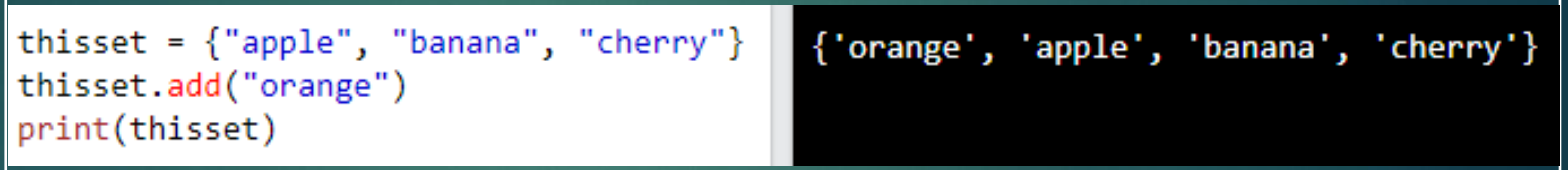
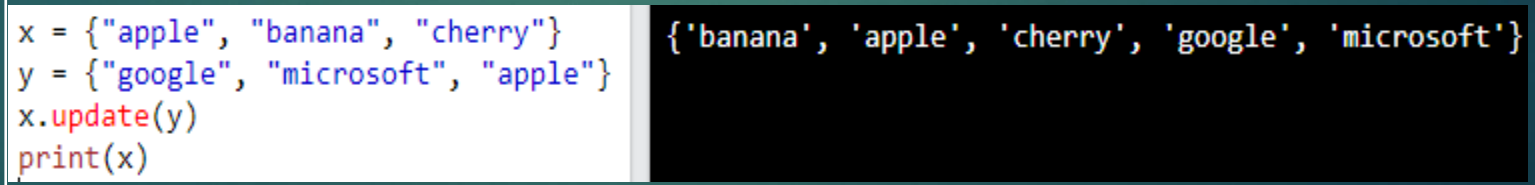
Sets

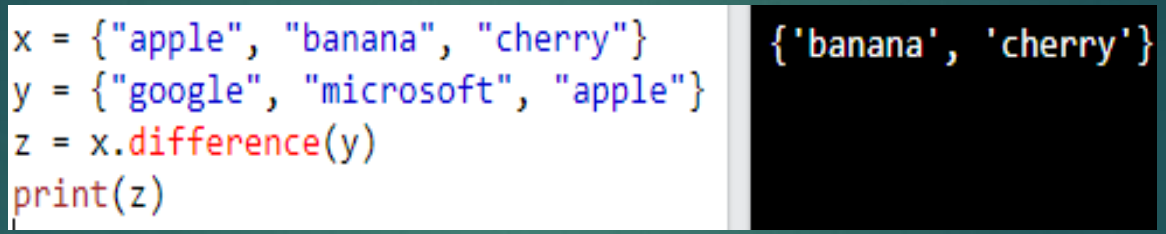
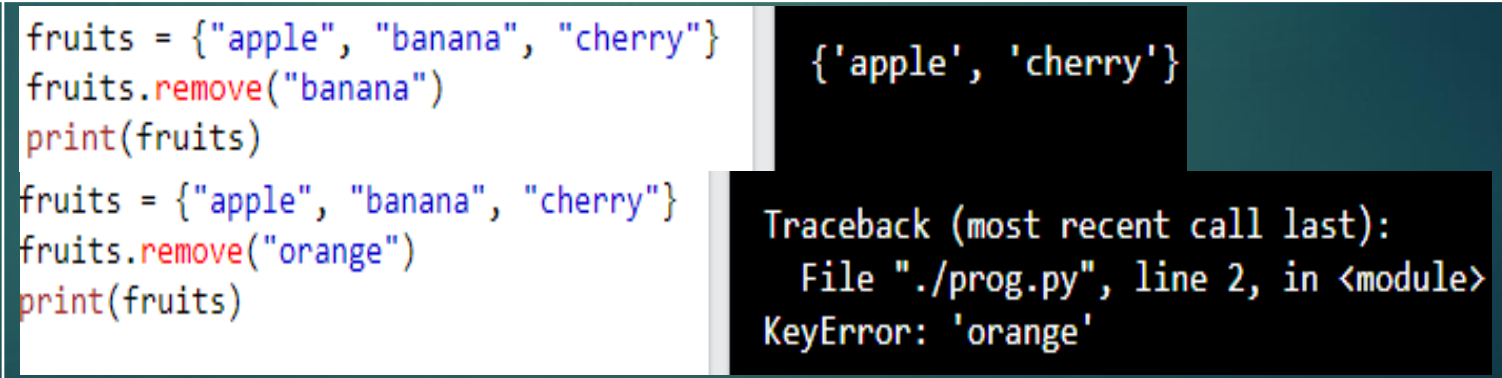
- Sets are used to store multiple items in a single variable.
- A set is a collection which is unordered, Unchangeable, unindexed and does not allows duplicate values.

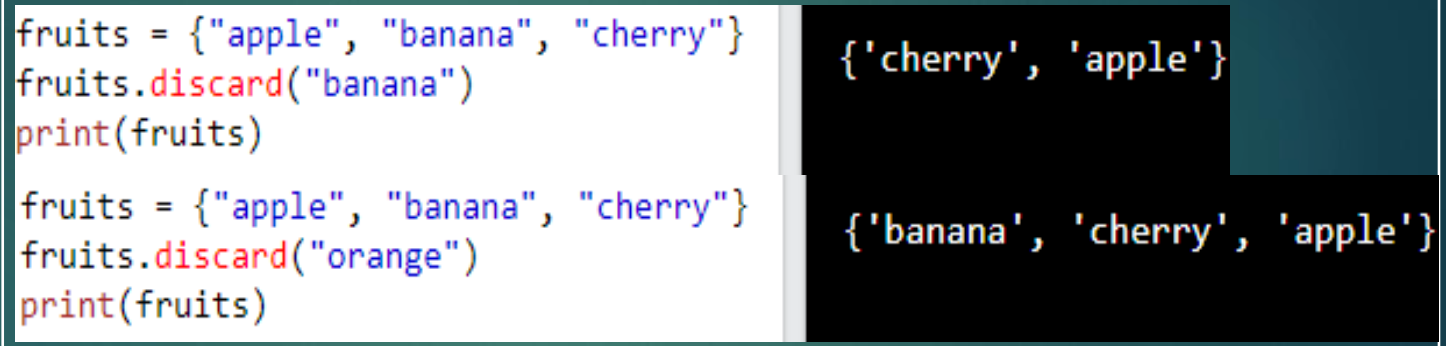
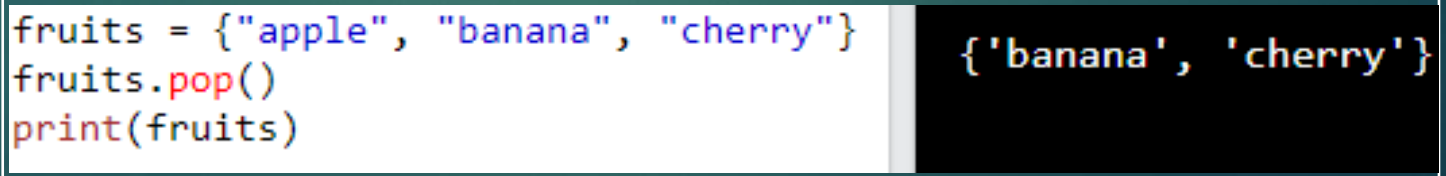
- Unordered means that the items in a set do not have a defined order.
- Unchangeable, meaning that we can change the items after the set has been created but we can add new items or remove the existing ones.
- Sets cannot have two items with the same value.

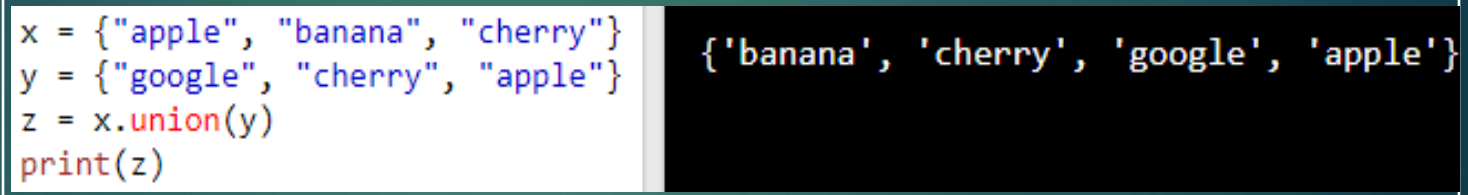
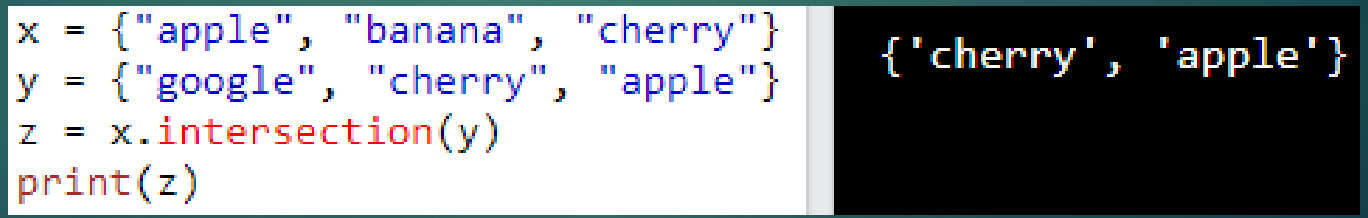
- Most of the method like len(), type() are same as discussed in the other data types.
- The Set() constructor are used to convert it into the set data type.

Method	Description
add()	Adds an element to the set
update()	Update the set with the union of this set and others
difference()	Returns a set containing the difference between two or more sets
remove()	Removes the specified element, Raise error if element does not exist in set
discard()	Remove the specified item, No error occur if item doesn't exist
pop()	Removes the first element from the set. *(No indexing allowed in it)*
union()	Return a set containing the union of sets
intersection()	Returns a set, that is the intersection of two other sets

Method	Example
add()	
update()	

Method	Example
difference()	
remove()	

Method	Description
discard()	
pop()	

Method	Description
union()	
intersection()	

Thank you!