3 Days Training on Python3

Day 3 : Module 10

Muhammad Saufy Rohmad

Module 10 (90 minutes)

Objectives
1. Collections, Tuples and Lists
4. Sets

1. Collections, Tuples and List

- Earlier in this book we looked at some Python built-in types such as string, int and float as well as bools
- These are not the only built-in types in Python; another group of built-in types are collectively known as collection types
- This is because they represent a collection of other types (such as a collection of strings, or integers).
- A collection is a single object representing a group of objects (such as a list or dictionary).
- Collections may also be referred to as containers (as they contain other objects).
- These collection classes are often used as the basis for more complex or application specific data structures and data types.
- These collection types support various types of data structures (such as lists and maps) and ways to process elements within those structures

1.1 Python Collection Types

- There are four classes in Python that provide container like behaviour; that is data types for holding collections of other objects, these are
  - Tuples
  - Lists
  - Sets
  - Dictionary

1.2 Tuples

- Tuples are defined using parentheses (i.e. round brackets '()') around the elements that make up the tuple, for example:

  tup1 = (1, 3, 5, 7)

- This means that a new tuple can be created from a Set, a List, a Dictionary (as these are all iterable types) or any type that implements the iterable protocol.

  list1 = [1, 2, 3]

  t1 = tuple(list1)

  print(t1)

1.2 Tuples(2)

- The elements of a Tuple can be accessed using an index in square brackets. The index returns the object at that position, for example:

print('tup1[0]:\t',tup1[0])

print('tup1[1]:\t',tup1[1])

print('tup1[2]:\t',tup1[2])

print('tup1[3]:\t',tup1[3])

1.2 Tuples(3)

- It is also possible to return what is known as a slice from a Tuple. This is a new Tuple which is comprised of a subset of the original Tuple.
- This is done by providing the start and end indexes for the slice, separated by a colon, within the index square brackets. For example:

print('tup1[1:3]:\t', tup1[1:3])

1.2 Tuples(4)

- Tuples can also contain a mixture of different types; that is they are not restricted to holding elements all of the same type. You can therefore write a Tuple such as:

  tup2 = (1, 'John', Person('Phoebe', 21), True, -23.45)

  print(tup2)

1.2 Tuples(5)

- You can iterate over the contents of a Tuple (that is process each element in the Tuple in turn).This is done using the for loop that we have already seen; however, it is the Tuple that is used for the value to which the loop variable will be applied:

```
tup3 = ('apple', 'pear', 'orange', 'plum', 'apple')
for x in tup3:
    print(x)
```

1.2 Tuples – Related Function

- print('len(tup3):\t', len(tup3))
- print(tup3.count('apple')) # returns 2
- print(tup3.index('pear')) # returns 1
- if 'orange' in tup3:

    print('orange is in the Tuple')

1.2 Nested Tuples

- Tuples can be nested within Tuples; that is a Tuple can contain, as one of its elements, another Tuple.

```
tuple1 = (1, 3, 5, 7)

tuple2 = ('John', 'Denise', 'Phoebe', 'Adam')

tuple3 = (42, tuple1, tuple2, 5.5)

print(tuple3)
```

1.3 Lists

- Lists are mutable ordered containers of other objects.
- They support all the features of the Tuple but as they are mutable it is also possible to add elements to a List remove elements and modify elements.
- The elements in the list maintain their order (until modified).

1.3 Lists(2)

- Creating Lists

  list1 = ['John', 'Paul', 'George', 'Ringo']

- Like tuples, we can also have nested lists

  l1 = [1, 43.5, Person('Phoebe', 21), True]

  l2 = ['apple', 'orange', 31]

  root_list = ['John', l1, l2, 'Denise']

  print(root_list)

- We can construct a list from a Tuple, a Dictionary or a Set.

1.3 Lists(3)

- Lists from tuple

  vowelTuple = ('a', 'e', 'i', 'o', 'u')

  print(list(vowelTuple))

- Assessing element in a list

  list1 = ['John', 'Paul', 'George', 'Ringo']

  print(list1[1])

- List Slicing

  list1 = ['John', 'Paul', 'George', 'Ringo']

  print('list1[1]:', list1[1])

  print('list1[-1]:', list1[-1])

  print('list1[1:3]:', list1[1:3])

  print('list[:3]:', list1[:3])

  print('list[1:]:', list1[1:])

1.3 Lists(4)

- Adding to the List

```
list1 = ['John', 'Paul', 'George', 'Ringo']
list1.append('Pete')
print(list1)
```

- Using extend()

```
list1 = ['John', 'Paul', 'George', 'Ringo', 'Pete']
print(list1)
list1.extend(['Albert', 'Bob'])
print(list1)
list1 += ['Ginger', 'Sporty']
print(list1)
```

1.3 Lists(5)

- Inserting into a List

  a_list = ['Adele', 'Madonna', 'Cher']

  print(a_list)

  a_list.insert(1, 'Paloma')

  print(a_list)

- List concatenation

  list1 = [3, 2, 1]

  list2 = [6, 5, 4]

  list3 = list1 + list2

  print(list3)

1.3 Lists(6)

- Removing from a List

  another_list = ['Gary', 'Mark', 'Robbie', 'Jason', 'Howard']

  print(another_list)

  another_list.remove('Robbie')

  print(another_list)

- Remove using pop()

  list6 = ['Once', 'Upon', 'a', 'Time']

  print(list6)

  print(list6.pop(2))

  print(list6)

1.3 Lists(7)

- Delete from a list

```
my_list = ['A', 'B', 'C', 'D', 'E']
print(my_list)
del my_list[2]
print(my_list)
```

- Delete a slice

```
my_list = ['A', 'B', 'C', 'D', 'E']
print(my_list)
del my_list[1:3]
print(my_list)
```

2. Sets

- In the last part we looked at Tuples and Lists; in this chapter we will look at a further container (or collection) types; the Set type.
- A Set is an unordered (unindexed) collection of immutable objects that does not allow duplicates.

2.1 Creating a Sets

- A Set is defined using curly brackets (e.g. ‘{}’).

  basket = {'apple', 'orange', 'apple', 'pear',

  'orange', 'banana'}

  print(basket)

- Set constructor function

  set1 = set((1, 2, 3)

  print(set1)

2.2 Assessing Element in a Set

- Elements of a Set can be iterated over using the for statement:

```
for item in basket:
    print(item)
```

- You can check for the presence of an element in a set using the in keyword

```
print('apple' in basket) #will print True is apple in the set basket
```

2.3 Adding Item to a Set

- It is possible to add items to a set using the add() method:

```
basket = {'apple', 'orange', 'banana'}
basket.add('apricot')
print(basket)
```

- If you want to add more than one item to a Set you can use the update() method:

```
basket = {'apple', 'orange', 'banana'}
basket.update(['apricot', 'mango', 'grapefruit'])
print(basket)
```

2.4 Other set operation

- It is not possible to change the items already in a Set
- You can find out the length of a Set using the len() function.

  basket = {'apple', 'orange', 'apple', 'pear', 'orange',

  'banana'}

  print(len(basket)) # generates 4

- You can also obtain the maximum or minimum values in a set using the max() and min() functions:

  print(max(a_set))

  print(min(a_set))

2.4 Other set operation(2)

- To remove an item from a set, use the remove() or discard() functions

  basket = {'apple', 'orange', 'apple', 'pear', 'orange',

  'banana'}

  print(basket)

  basket.remove('apple')

  print(basket)

- The method clear() is used to remove all elements from a Set:

  basket = {'apple', 'orange', 'banana'}

  basket.clear()

  print(basket)

2.4 Other set operation(3)

- The Set container also supports set like operations such as (|), intersection (&),difference (−) and symmetric difference (^). These are based on simple Set theory.

print('Union:', s1 | s2)

s1 = {'apple', 'orange', 'banana'}

s2 = {'grapefruit', 'lime', 'banana'}

Union: {'apple', 'lime', 'banana', 'grapefruit', 'orange'}

2.4 Other set operation(4)

- The intersection of two sets represents the common values between two sets:

  print('Intersection:', s1 & s2)

  Intersection: {'banana'}

- The difference between two sets is the set of values in the first set that are not in the second set:

  print('Difference:', s1 – s2)

  Difference: {'apple', 'orange'}

2.4 Other set operation(5)

- The symmetric difference represents all the unique values in the two sets (that is it is the inverse of the intersection:

  print('Symmetric Difference:', s1 ^ s2)

- The output is

  Symmetric Difference: {'orange', 'apple', 'lime', 'grapefruit'}