CORVIT
Reaching end to end

Introducing Python

Lecture# 5

by

python™

Umair bin Mansoor

Network Programming Planner

OBJECTIVES

After this session, students will be able to:

- **Create and use tuples as fixed elements from being added, deleted or replaced**
- **Apply common sequence operations for tuples**
- **Create sets and to add and remove elements in a set**
- **Use the `len`, `min`, `max`, and `sum` functions and to use the `in` and `not in` operators to determine whether an element is in a set**
- **Create dictionaries and to add, modify, and retrieve elements in a dictionary using the syntax `dictionaryName[key]`**
- **Delete items in a dictionary using the `del` keyword**
- **Obtain the size of a dictionary using the `len` function**
- **Test whether a key is in a dictionary using the `in` or `not in` operator**
- **Use the `keys`, `values`, `items`, `clean`, `get`, `pop`, and `popitem` methods on a dictionary**

TUPLES

- **Tuples are like lists, but their elements are fixed; that is, once a tuple is created, you cannot add new elements, delete elements, replace elements, or reorder the elements in the tuple.**
- **You create a tuple by enclosing its elements inside a pair of parentheses. The elements are separated by commas. You can create an empty tuple and create a tuple from a list, as shown in the following example:**
  - `t1 = ()` *# Create an empty tuple*
  - `t2 = (1,3,5)` *# Create a tuple with three elements*
  - `t3 = tuple([1,2,3,4,5])` *# Create a tuple from a list*
  - `t4 = tuple("abac")` *# t4 is ['a', 'b', 'a', 'c']*
  - `t5 = [2, "three", 4.0]` *# Mixed data-type elements*
- **All the basic operations of sequences mentioned in lecture #4 can be applied on tuples**

python™

TupleDemo(5_1).py

```
1. tuple1 = ("green", "red", "blue") # Create a tuple
2. print(tuple1)
3. tuple2 = tuple([7, 1, 2, 23, 4, 5]) # Create a tuple from a list
4. print(tuple2)
5. print("length is", len(tuple2)) # Use function len
6. print("max is", max(tuple2)) # Use max
7. print("min is", min(tuple2)) # Use min
8. print("sum is", sum(tuple2)) # Use sum
9. print("The first element is", tuple2[0]) # Use index operator
10.tuple3 = tuple1 + tuple2 # Combine two tuples
11.print(tuple3)
12.tuple3 = 2 * tuple1 # Duplicate a tuple
13.print(tuple3)
14.print(tuple2[2 : 4]) # Slicing operator
```

TupleDemo(5_1).py (Contd.)

```
16.print(tuple1[-1])
17.print(2 in tuple2) # in operator
18.list1 = list(tuple2) # Obtain a list from a tuple
19.list1.sort()
20.tuple4 = tuple(list1)
21.tuple5 = tuple(list1)
22.print(tuple4)
23.print(tuple4 == tuple5) # Compare two tuples
```

SETS

- **Sets are like lists in that you use them for storing a collection of elements. Unlike lists, however, the elements in a set are non-duplicates and are not placed in any particular order.**
- **You can create a set of elements by enclosing the elements inside a pair of curly braces ({}). The elements are separated by commas. You can create an empty set, or you can create a set from a list or a tuple, as shown in the following examples:**
  - `s1 = set()` *# Create an empty set*
  - `s2 = {1,3,5}` *# Create a set with three elements*
  - `s3 = set((1,3,5))` *# Create a set from a tuple*
  - `s4 = set([1,3,5])` *# Create a set from a list*
  - `s5 = set("abac")` *# s5 is {'a','b','c'} # Create a set from a string*
  - `s6 = {2, "three", 4.0}` *# Mixed data-type elements*
- **All the basic operations of sequences mentioned in lecture #4 can be applied on sets**

SET OPERATIONS

- You can add an element to a set or remove an element by using the `add(e)` or `remove(e)` method.
- A set `s1` is a subset of `s2` if every element in `s1` is also in `s2`. You can use the `s1.issubset(s2)` method to determine whether `s1` is a subset of `s2`.
- A set `s1` is a superset of set `s2` if every element in `s2` is also in `s1`. You can use the `s1.issuperset(s2)` method to determine whether `s1` is a superset of `s2`.
- You can use the == and != operators to test if two sets contain the same elements. For example:

```
>>> s1 = {1, 2, 4}
>>> s2 = {1, 4, 2}
>>> s1 == s2
True
>>> s1 != s2
False
```

`s1 < s2` returns `True` if `s1` is a proper subset of `s2`.

`s1 <= s2` returns `True` if `s1` is a subset of `s2`.

`s1 > s2` returns `True` if `s1` is a proper superset of `s2`.

`s1 >= s2` returns `True` if `s1` is a superset of `s2`.

SET OPERATIONS

- Python provides the methods for performing set *union*, *intersection*, *difference*, and *symmetric difference* operations.
- The union of two sets is a set that contains all the elements from both sets. You can use the `union` method or the | operator to perform this operation. For example:

```
>>> s1 = {1, 2, 4}
>>> s2 = {1, 3, 5}
>>> s1.union(s2)
{1, 2, 3, 4, 5}
>>>
>>> s1 | s2
{1, 2, 3, 4, 5}
>>>
```

SET OPERATIONS

- **The intersection of two sets is a set that contains the elements that appear in both sets. You can use the `intersection` method or the `&` operator to perform this operation. For example:**

```
>>> s1 = {1, 2, 4}
>>> s2 = {1, 3, 5}
>>> s1.intersection(s2)
{1}
>>>
>>> s1 & s2
{1}
>>>
```

SET OPERATIONS

- **The difference between set1 and set2 is a set that contains the elements in set1 but not in set2. You can use the `difference` method or the – operator to perform this operation. For example:**

```
>>> s1 = {1, 2, 4}
>>> s2 = {1, 3, 5}
>>> s1.difference(s2)
```

```
{2, 4}
>>>
>>> s1 - s2
{2, 4}
>>>
```

SET OPERATIONS

- The symmetric difference (or exclusive or) of two sets is a set that contains the elements in either set, but not in both sets. You can use the `symmetric_difference` method or the ^ operator to perform this operation. For example:

```
>>> s1 = {1, 2, 4}
>>> s2 = {1, 3, 5}
>>> s1.symmetric_difference(s2)
{2, 3, 4, 5}
>>>
>>> s1 ^ s2
{2, 3, 4, 5}
>>>
```

SetDemo(5_2).py

```
1. set1 = {"green", "red", "blue", "red"} # Create a set
2. print(set1)
3. set2 = set([7, 1, 2, 23, 2, 4, 5]) # Create a set from a list
4. print(set2)
5. print("Is red in set1?", "red" in set1)
6. print("length is", len(set2)) # Use function len
7. print("max is", max(set2)) # Use max
8. print("min is", min(set2)) # Use min
9. print("sum is", sum(set2)) # Use sum
10.set3 = set1 | {"green", "yellow"} # set3 = set1.union({"green","yellow"})
11.print(set3)
12.set3 = set1 - {"green", "yellow"} # set3 = set1.difference({"green","yellow"})
13.print(set3)
14.set3 = set1 & {"green", "yellow"} # set3 = set1.intersection({"green","yellow"})
15.print(set3)
```

SetDemo(5_2).py (Contd.)

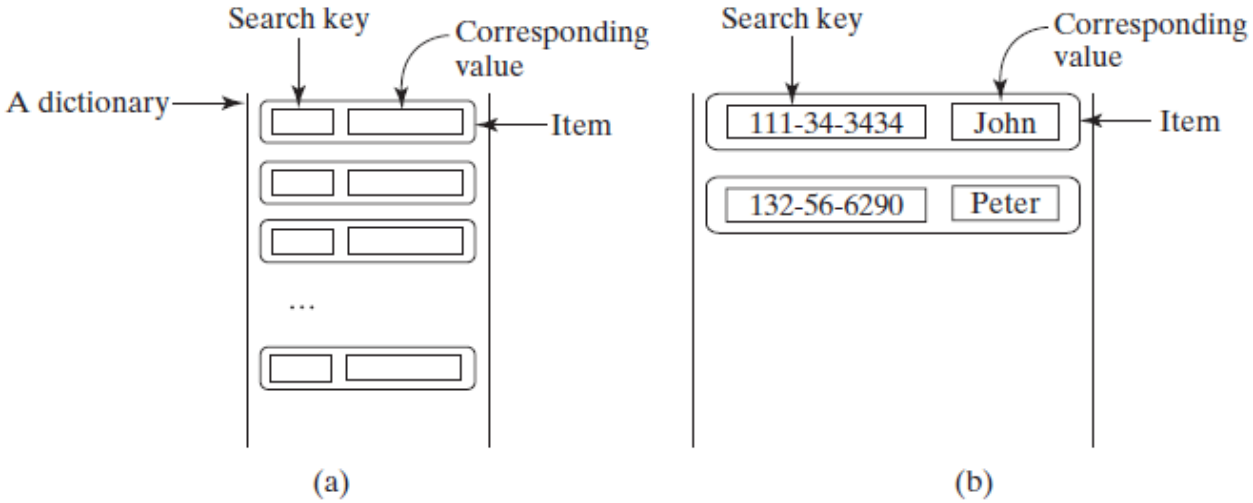
```
16.set3 = set1 ^ {"green", "yellow"} # set3 = set1. symmetric_difference({"green","yellow"})
17.print(set3)
18.list1 = list(set2) # Obtain a list from a set
19.print(set1 == {"green", "red", "blue"}) # Compare two sets
20.set1.add("yellow")
21.print(set1)
22.set1.remove("yellow")
23.print(set1)
```

Comparing the Performance of Sets and Lists

- Sets are more efficient than lists for the `in` and `not in` operator and for the remove method.
- The elements in a list can be accessed using the index operator.
- However, sets do not support the index operator, because the elements in a set are unordered.
- We now conduct an interesting experiment to test the performance of sets and lists.
- The program SetListPerformanceTest.py compares the execution time of the operations on Lists and Sets

Dictionaries

- A dictionary is a container object that stores a collection of key/value pairs. It enables fast retrieval, deletion, and updating of the value by using the key.
- The data structure is a called a "dictionary" because it resembles a word dictionary, where the words are the keys and the words' definitions are the values.
- A dictionary is also known as a map, which maps each key to a value.



Creating a Dictionary

- You can create a dictionary by enclosing the items inside a pair of curly braces ({}).
- Each item consists of a key, followed by a colon, followed by a value.
- The items are separated by commas. For example, the following statement:
  - `student = {"111-34-3434":"John"}` *# One item dictionary*
  - `students = {"111-34-3434":"John", "132-56-6290":"Peter"}`
  - `Names = {“First_Name":["John",“Susan":"Peter“]}` *# One item dictionary*
  - `student = {}` *# empty dictionary*
- To add an item to a dictionary, use the syntax:
  - `dictionaryName[key] = value`
- For Example:
  - `students["234-56-9010"] = "Susan“`
- If the key is already in the dictionary, the preceding statement replaces the value for the key.
- `Val = students["234-56-9010"]` *# Val = “Susan”*

The Dictionary Methods

dict	
keys(): tuple	Returns a sequence of keys.
values(): tuple	Returns a sequence of values.
items(): tuple	Returns a sequence of tuples. Each tuple is (key, value) for an item.
clear(): None	Deletes all entries.
get(key): value	Returns the value for the key.
pop(key): value	Removes the item for the key and returns its value.
popitem(): tuple	Returns a randomly selected key/value pair as a tuple and removes the selected item.

The Dictionary Methods Example

```
>>> students = {"111-34-3434":"John", "132-56-6290":"Peter"}
>>> tuple(students.keys())
("111-34-3434", "132-56-6290")
>>> tuple(students.values())
("John", "Peter")
>>> tuple(students.items())
(("111-34-3434", "John"), ("132-56-6290", "Peter"))
>>> students.get("111-34-3434")
"John"
>>> print(students.get("999-34-3434"))
None
>>> students.pop("111-34-3434")
"John"
>>> students
{"132-56-6290":"Peter"}
>>> students.clear()
>>> students
{}
>>>
```

Questions & Answers