# PYTHON PROGRAMMING - I

## Chap - 4

# Python Tuples And Dictionary

By-

Prof. A. P. Chaudhari

(M.Sc. Computer Science, SET)

HOD,

Department of Computer Science

S.V.S's Dadasaheb Rawal College,

Dondaicha

## Creating Tuples:

Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values between parentheses.

For example −

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
```

The empty tuple is written as two parentheses containing nothing −

```
tup1 = ();
```

To write a tuple containing a single value you have to include a comma, even though there is only one value −

```
tup1 = (50,);
```

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

## Accessing Elements in Tuples:

To access elements in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index.

For example −

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5, 6, 7 );
print "tup1[0]: ", tup1[0]
print "tup2[1:5]: ", tup2[1:5]
```

When the above code is executed, it produces the following result −

```
tup1[0]:  physics
tup2[1:5]:  [2, 3, 4, 5]
```

**Updating Tuples:**

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates −

```
tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');

# tup1[0] = 100;                    # This action is not valid for tuples

# So let's create a new tuple as follows
tup3 = tup1 + tup2;
print tup3
```

When the above code is executed, it produces the following result −

```
(12, 34.56, 'abc', 'xyz')
```

**Delete Tuple Elements:**

Removing individual tuple elements is not possible. To explicitly remove an entire tuple, just use the **del** statement. For example −

```
tup = ('physics', 'chemistry', 1997, 2000);
print tup
del tup;
print "After deleting tup : “, tup
```

This produces the following result. Note an exception raised, this is because after **del tup** tuple does not exist any more −

```
('physics', 'chemistry', 1997, 2000)
Traceback (most recent call last):
   File "test.py", line 8, in <module>
 print "After deleting tup : “, tup;
NameError: name 'tup' is not defined
```

## Basic Tuples Operations:

Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

In fact, tuples respond to all of the general sequence operations we used on strings.

| Python Expression | Results | Description |
|---|---|---|
| len(1,2,3) | 3 | Length |
| (1,2,3) + (4,5,6) | (1,2,3,4,5,6) | Concatenation |
| ('Hi!',) * 4 | ('Hi!','Hi!','Hi!','Hi!') | Repetition |
| 3 in (1,2,3) | True | Membership |
| for x in (1,2,3): print x, | 1 2 3 | Iteration |

## Built-in Tuple Functions:

1) **cmp():** Python tuple method **cmp()** compares elements of two tuples.

**Syntax:** **cmp(a, b)**

Parameters: a and b are two tuples in which the comparison is being done.

Returns: **-1** if a<b, **0** if a=b **1** if a>b

```
e.g.1:    tuple1 = (1,2,3)
          tuple2 = (4,5,6)
          print cmp(tuple1, tuple2)          O/P: -1
e.g.2:    tuple1 = (1,2,3)
          tuple2 = (4,5,6)
          print cmp(tuple2, tuple1)          O/P: 1
e.g.3:    tuple1 = ('abc','mno')
          tuple2 = ('abc','mno')
          print cmp(tuple1, tuple2)          O/P: 0
```

## Built-in Tuple Functions:

**2) len():** Python tuple method **len()** returns the number of elements in the *tuple*.

**Syntax:** **len(tuple)**

Parameters: tuple − This is a tuple for which number of elements to be counted.

Returns: This method returns the number of elements in the tuple.

e.g.:

```
tuple1 = (101, ‘Atharv’, ‘Dhule’, 30000)
tuple2 = (‘English’, ‘Maths’, ‘Science’)
print “Number of elements in tuple1:”, len(tuple1)

print “Number of elements in tuple2:”, len(tuple2)
```

O/P:

```
Number of elements in tuple1: 4
Number of elements in tuple2: 3
```

## Built-in Tuple Functions:

**3) max():** Python tuple method **max()** returns the elements from the *tuple* with maximum value.

**Syntax:** **max(tuple)**

Parameters: tuple − This is a tuple from which maximum valued element to be returned.

Returns: This method returns the elements from the tuple with maximum value.

e.g.:

```
tuple1 = (400, 100, 700, 300)
tuple2 = (‘Maths’, ‘English’, ‘Science’)
print “Maximum element in tuple1:”, max(tuple1)

print “Maximum element in tuple2:”, max(tuple2)
```

O/P:

```
Maximum element in tuple1: 700
Maximum element in tuple2:: Science
```

## Built-in Tuple Functions:

**4) min():** Python tuple method **min()** returns the elements from the *tuple* with minimum value.

**Syntax:** **min(tuple)**

Parameters: tuple − This is a tuple from which min valued element to be returned.

Returns: This method returns the elements from the tuple with minimum value.

e.g.:
```
tuple1 = (400, 100, 700, 300)
tuple2 = (‘Maths’, ‘English’, ‘Science’)
print “Minimum element in tuple1:”, min(tuple1)
print “Minimum element in tuple2:”, min(tuple2)
```

O/P:
```
Minimum element in tuple1: 100
Minimum element in tuple2:: English
```

## Built-in Tuple Functions:

**5) tuple():** Python tuple method **tuple()** takes sequence types and converts them to tuples. This is used to convert a given list into tuple.

**Syntax:** **tuple(seq)**

Parameters: **seq** − This is a list to be converted into tuple.

Returns: This method returns the tuple.

e.g.:
```
list1 = [101, 'Atharv', 'Dhule', 30000]
tuple1 = tuple(list1)
print “Tuple Elements: ", tuple1
```

O/P:
```
Tuple Elements:  (101, 'Atharv', 'Dhule', 30000)
```

## Built-in Tuple Functions:

**6) any():** Python tuple method **any()** returns true if any element present in a tuple and return false if tuple is empty.

**Syntax:** **any(tuple)**

e.g.:

```
tuple1 = (400, 100, 700, 300)
tuple2 = ()
print “Is there any element in Tuple1: ", any(tuple1)
print “Is there any element in Tuple2: ", any(tuple2)
```

O/P:

```
Is there any element in Tuple1: True
Is there any element in Tuple2: False
```

## Built-in Tuple Functions:

**7) sorted():** Python tuple method **sorted()** is used to sort all elements of the tuple.

**Syntax:** **sorted(tuple)**

e.g.:

```
tuple1 = (400, 100, 700, 300)
print “Original Tuple: “, tuple1
print “Sorted Tuple: ", sorted(tuple1)
```

O/P:

```
Original Tuple:  (400, 100, 700, 300)
Sorted Tuple:  (100, 300, 400, 700)
```

## Built-in Tuple Functions:

**8) sum():** Python tuple method **sum()** returns sum of all elements of the tuple.

**Syntax:** **sum(tuple)**

e.g.:
```
tuple1 = (400, 100, 700, 300)
print “Sum of Tuple Elements: “, sum(tuple1)
```

O/P:
```
Sum of Tuple Elements: 1500
```

## Built-in Tuple Methods:

**9) count():** Python tuple method **count()** returns the number of times element occur in a tuple, means count() method searches the given element in a tuple and return how many times the element occurred in tuple.

**Syntax:** **tuple.count(element)**

e.g.:

```
tuple1 = (101, 'Atharv', 'Dhule', 101, 30000)
print "Count for 101: ",tuple1.count(101)
print "Count for Atharv: ",tuple1.count('Atharv')
```

O/P:

```
Count for 101:  2
Count for Atharv:  1
```

## Built-in Tuple Methods:

**10) index():** Python tuple method **index()** searches an element in a tuple and returns its index. This method finds whether the given element present is tuple, if element is present it returns its position and if not present it returns 0. However, if the same element is present more than once, the first/smallest position is returned.

**Syntax:** **tuple.index(element)**

e.g.:

```
tuple1 = (101, 'Atharv', 'Dhule', 101, 30000)
tuple2 = ('English', 'Maths', 'Science')
print "Index for 101: ",tuple1.index(101)
print "Index for Science: ", tuple2.index('Science')
```

O/P:

```
Index for 101:  0
Index for Science:  2
```

## Introduction to Dictionary:

- Python's dictionaries are kind of hash table type.
- They work like associative arrays and consist of **key-value pairs**.
- A dictionary **key** can be almost any Python type, but are usually numbers or strings.
- **Values** on the other hand, can be of any type.
- Each key is separated from its value by a colon (:).
- The items are separated by commas.
- The whole thing is enclosed in curly braces.
- An empty dictionary without any items is written with just two curly braces, like this: {}.
- Keys are unique within a dictionary while values may not be.
- E.g.: dict1 = {‘Name': ‘Parth', ‘Age':11, ‘Class': ‘Fifth'}
- Dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

**Accessing Values in Dictionary:**

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example −

```
dict1 = {‘Name': ‘Parth', ‘Age':11, ‘Class': ‘Fifth'}
print "dict1['Name']: ", dict1['Name']
print "dict1['Age']: ", dict1['Age']
```

```
O/P:    dict1['Name']:  Parth
        dict1['Age']:  11
```

If we attempt to access a data item with a key, which is not part of the dictionary, we get an error as follows −

```
dict1 = {‘Name': ‘Parth', ‘Age':11, ‘Class': ‘Fifth'}
print "dict1['Address']: ", dict1[‘Address']
```

```
O/P:    dict1['Address]:
        Traceback (most recent call last):   File "test.py", line 4, in <module>
            print "dict1['Address']: ", dict1['Address'];  KeyError: 'Address'
```

**Updating Dictionary:**

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example −

```
dict1 = {‘Name': ‘Parth', ‘Age':11, ‘Class': ‘Fifth'}
dict1['Age'] = 12;                    # update existing entry
dict1['School'] = “Rotary School";          # Add new entry
print "dict1['Age']: ", dict1['Age']
print "dict1['School']: ", dict1['School']
```

When the above code is executed, it produces the following result −

```
dict1['Age']:  12
dict1['School']:  Rotary School
```

**Delete Dictionary Elements:**

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the **del** statement.

e.g.:

```
dict1 = {‘Name': ‘Parth', ‘Age':11, ‘Class': ‘Fifth'}
del dict1['Name'];              # remove entry with key 'Name'
dict1.clear();                  # remove all entries in dict
del dict1 ;                     # delete entire dictionary
print "dict1['Age']: ", dict1['Age']
print "dict1['School']: ", dict1['School']
```

This produces an exception is raised because after **del dict** dictionary does not exist any more.

## Properties of Dictionary Keys:

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys −

**a)** More than one entry per key not allowed. Which means no duplicate key is allowed.

e.g.:
```
dict1 = {‘Name': ‘Parth', ‘Age':11, ‘Name': ‘Rohit'}
print "dict1['Name']: ", dict1['Name']
```

O/P:
```
dict1['Name']:  Rohit
```

**b)** Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

e.g.:
```
dict1 = {['Name']: ‘Parth', 'Age': 11}
print "dict1['Name']: ", dict1['Name']
```

O/P: TypeError:

list objects are unhashable

## Built-in Dictionary Functions:

1) **cmp():** Python dictionary method **cmp()** compares elements of two dictionaries.

**Syntax:** **cmp(a, b)**

Parameters: a and b are two dictionaries in which the comparison is being done.

Returns: **-1** if a<b, **0** if a=b **1** if a>b

e.g.:

```
dict1 = {"Name":"Kajal","Age":10}
dict2 = {"Name":"Harsha","Age":15}
dict3 = {"Name":"Divya","Age":8}
dict4 = {"Name":"Harsha","Age":15}
print cmp(dict1,dict2)          O/P:    -1
print cmp(dict2,dict3)          O/P:    1
print cmp(dict2,dict4)          O/P:    0
```

## Built-in Dictionary Functions:

**2) len():** Python dictionary method **len()** returns the number of elements in the *dictionary*.

**Syntax:** **len(dictionary)**

Parameters: dictionary − This is a dictionary for which number of elements to be counted.

Returns: This method returns the number of elements in the dictionary.

e.g.:

```
dict1 = {"Name":"Kajal","Age":10}
dict2 = {"Name":"Harsha","Age":15 ,"City":"Mumbai"}
print “Number of elements in dict1: ”, len(dict1)
print “Number of elements in dict2: ”, len(dict2)
```

O/P:

```
Number of elements in dict1: 2
Number of elements in dict2: 3
```

## Built-in Dictionary Functions:

**3) str():** Python dictionary method **str()** produces a printable string representation of a dictionary.

**Syntax:** **str(dictionary)**

Returns: This method returns string representation.

e.g.:
```
dict1 = {"Name":"Kajal","Age":10}
print “Dictionary in string format: ”, str(dict1)
```

O/P:
```
Dictionary in string format:  {'Age': 10, 'Name': 'Kajal'}
```

## Built-in Dictionary Functions:

**4) type():** Python dictionary method **type()** returns the type of the passed variable. If passed variable is dictionary then it would return a dictionary type.

**Syntax:** **type(dictionary)**

Returns: This method returns the type of the passed variable.

e.g.:

```
d1 = {"Name":"Kajal","Age":10}
l1 = ["Komal",15]
s1 = "Pooja"
print type(d1)          O/P:    <type 'dict'>
print type(l1)          O/P:    <type 'list'>
print type(s1)          O/P:    <type 'str'>
```

## Built-in Dictionary Methods:

**1) clear():** Python dictionary method **clear()** removes all items from the dictionary.

**Syntax:** **dict.clear()**

Returns: This method does not return any value.

e.g.:

```
dict1 = {"Name":"Harsha","Age":15 ,"City":"Mumbai"}
print “Number of elements in dict1: ”, len(dict1)
dict1.clear()
print “Number of elements in dict1: ”, len(dict1)
```

O/P:

```
Number of elements in dict1: 3
Number of elements in dict1: 0
```

## Built-in Dictionary Methods:

**2) copy():** Python dictionary method **copy()** returns a copy of the dictionary.

**Syntax:** **dict.copy()**

Returns: This method returns a copy of the dictionary.

e.g.:

```
dict1 = {"Name":"Harsha","Age":15 ,"City":"Mumbai"}
dict2 = dict1.copy()
print “Elements of dict2: ”, dict2
```

O/P:

```
Elements of dict2: {"Name":"Harsha","Age":15 ,"City":"Mumbai"}
```

## Built-in Dictionary Methods:

**3) fromkeys():** Python dictionary method **fromkeys()** creates a new dictionary with keys from *seq* and *values* set to value.

**Syntax:** **dict. fromkeys(seq[, value])**

Parameters: **seq** − This is the list of values which would be used for dictionary keys preparation.

**value** − This is optional, if provided then value would be set to this value.

Returns: This method returns the list.

e.g.:

```
seq = ('Name', 'Age', 'Class')
dict = dict.fromkeys(seq)
print "New Dictionary :", dict
dict = dict.fromkeys(seq, 10)
print "New Dictionary :", dict
```

```
O/P:
New Dictionary :
{'Age': None, 'Name': None, 'Class': None}
New Dictionary :
{'Age': 10, 'Name': 10, 'Class': 10}
```

## Built-in Dictionary Methods:

**4) get():** Python dictionary method **get()** returns a value for the given key. If key is not available then returns default value None.

**Syntax:** **dict.get(key, default = None)**

Parameters: **key** − This is the Key to be searched in the dictionary.

**default** − This is the Value to be returned in case key does not exist.

Returns: This method return a value for the given key. If key is not available, then returns default value None.

e.g.:

```
dict1 = {'Name':'Harsha','Age':15 ,'City':'Mumbai'}
print 'Name value is:',dict1.get('Name')
print 'Class value is:',dict1.get('Class',10)
```

O/P:

```
Name value is: Harsha
Class value is: 10
```

## Built-in Dictionary Methods:

**5) has_key():** Python dictionary method **has_key()** returns true if a given *key* is available in the dictionary, otherwise it returns a false.

**Syntax:** **dict.has_key(key)**

Parameters: **key** − This is the Key to be searched in the dictionary.

Returns: This method return true if a given key is available in the dictionary, otherwise it returns a false.

e.g.:
```
dict1 = {'Name':'Harsha','Age':15 ,'City':'Mumbai'}
print 'Name key available:',dict1.has_key('Name')
print 'Class key available:',dict1.has_key('Class')
```

O/P:
```
Name key available: True
Class key available: False
```

## Built-in Dictionary Methods:

**6) items():** Python dictionary method **items()** returns a list of dict's (key, value) tuple pairs.

**Syntax:** **dict.items()**

Returns: This method returns a list of tuple pairs.

e.g.:

```
dict1 = {'Name':'Harsha','Age':15 ,'City':'Mumbai'}
print 'Elements in dict1:',dict1.items()
```

O/P:

```
Elements in dict1: [('City', 'Mumbai'), ('Age', 15), ('Name', 'Harsha')]
```

## Built-in Dictionary Methods:

**7) keys():** Python dictionary method **keys()** returns a list of all the available keys in the dictionary.

**Syntax:** **dict.keys()**

Returns: This method returns a list of all the available keys in the dictionary.

e.g.:
```
dict1 = {'Name':'Harsha','Age':15 ,'City':'Mumbai'}
print 'Keys in dict1:',dict1.keys()
```

O/P:
```
Keys in dict1: ['City', 'Age', 'Name']
```

# Built-in Dictionary Methods:

**8) values():** Python dictionary method **values()** returns a list of all the values available in a given dictionary.

**Syntax:** **dict.values()**

Returns: This method returns a list of all the values available in a given dictionary.

e.g.:
```
dict1 = {'Name':'Harsha','Age':15 ,'City':'Mumbai'}
print 'Keys in dict1:',dict1.values()
```

O/P:
```
Keys in dict1: ['Mumbai', 15, 'Harsha']
```

## Built-in Dictionary Methods:

**9) update():** Python dictionary method **update()** adds dictionary dict2's key-values pairs in to dict1. This function does not return anything.

**Syntax:** **dict1.update(dict2)**

Parameters: **dict2** − This is the dictionary to be added into dict1.

Returns: This method does not return any value.

e.g.:

```
dict1 = {'Name':'Harsha','Age':15}
dict2 = {'Class':10}
dict1.update(dict2)
print 'Updated Dictionary:',dict1
```

O/P:

```
Updated Dictionary: {'Age': 15, 'Name': 'Harsha', 'Class': 10}
```