Advanced Programming-Python

Presented by

Siddhanta Borah

Contents of Todays (Day-8) Discussion

- Operations of Tuples
  - Creating Tuples
  - Accessing elements and slicing
  - Immutable nature
  - Features of Tuple- Tuple Assignment, Tuple as return value
  - Basic Tuple Operations
  - Built in Tuple functions
- Operations of Dictionaries
  - Creating a Dictionary
  - Accessing element
  - Updating Dictionary
  - Deleting elements from dictionary
  - Operations in Dictionary
  - Build in dictionary methods

CREATING TUPLES

Creating tuples is easy in Python. In order to create a tuple, all the items or elements are placed inside parentheses separated by commas and assigned to a variable. The parentheses, at the time of creating a tuple, is not necessary, but it is a good practice to use parentheses. Tuples can have any number of different data items (that is, integer, float, string, list, etc.).

Examples

1. A tuple with integer data items

```
>>> tuple = (4,2,9,1)
>>> print tuple
(4,2,9,1)          # Output
```

CREATING TUPLES

2. A tuple with items of different data types

```
>>>tuple_mix = (2,30,"Python",5.8,"Program")
>>>print tuple_mix
(2, 30, 'Python', 5.8, 'Program')            # Output
```

3. Nested tuple

```
>>>nested_tuple = ("Python", [1,4,2],["john",3.9])
>>> print nested_tuple
('Python', [1, 4, 2], ['john', 3.9])         # Output
```

4. Tuple can also be created without parenthesis

```
>>>tuple =4.9,6,'house'
>>>print tuple
(4.9, 6, 'house')          # Output
```

ACCESSING VALUES IN TUPLES

In order to access the values in a tuple, it is necessary to use the index number enclosed in square brackets along with the name of the tuple.

Example 1: Using square brackets

```
>>>tup1 = ('Physics','chemistry','mathematics')
>>>tup2 = (10,20,30,40,50)
>>>print tup1[1]
Chemistry              # Output
>>>print tup2[4]
50                     # Output
>>>print tup1[0]
Physics                # Output
>>> print tup2[0]
10                     # Output
```

ACCESSING VALUES IN TUPLES USING SLICING

```
>>>tup1 = ('Physics','chemistry','mathematics')
>>>tup2 = (10,20,30,40,50)
>>>tup2[1:4]
(20, 30, 40)                    # Output
>>>tup1[:1]
('Physics',)                    # Output
>>>tup1[:2]
('Physics', 'chemistry')        # Output
```

IMMUTABLE NATURE OF TUPLE

Tuples are immutable. The values or items in the tuple cannot be changed once it is declared. If we want to change the values, we have to create a new tuple.

Example

```
# declaring a tuple
>>>tup = (12, 15, "Python", 2.3)
# change the 3rd element "Python" to "Hello"
>>>tup[2] = "Hello"
TypeError: 'tuple' object does not support item assignment
```

In the given example, a tuple `tup` is declared with some items. Now, if we try changing the 3rd element `"Python"` to `"Hello"` using the assignment operator, the Python interpreter generates an error. From this example, it is clear that the values of tuple elements cannot be changed afterwards.

FEATURES OF TUPLE- TUPLE ASSIGNMENT

Tuple assignment is a very attractive and powerful feature in Python. It allows the assignment of values to a tuple of variables on the left side of the assignment from the tuple of values on the right side of the assignment.

The number of variables in the tuple on the left of the assignment must match the number of elements/ items in the tuple on the right of the assignment.

Example

```
# creating a tuple
>>>Anil = ('221','Anil','Rahul','Delhi',1971,'Jaipur Gwalior')
# tuple assignment
>>>(id,fst_name,lst_name,city,year_of_birth,birth_place) = Anil
```

FEATURES OF TUPLE- TUPLE ASSIGNMENT

Here, we created a tuple named `john`, with 7 elements inside it. Now, in the next statement, the value of each element of this tuple is assigned to the respective variables. It can be seen that the number of variables to the left of assignment is seven and the number of items in the tuple is also seven; hence, the number of values are matching and the assignment is successful.

Now,

```
>>>print id
221          # Output
>>>print fst_name
John         # Output
>>>print year_of_birth
1971         # Output
>>>print birth_place
atlanta Georgia          # Output
```

In the assignment statement, each variable is assigned with a value that was inside the tuple and can be accessed individually. If we had used the traditional assignment procedure, it would have been done in 7 lines of statement. With the help of tuple assignment, it is done in a one-line statement.

FEATURES OF TUPLE- TUPLE ASSIGNMENT

Similarly, sometimes we need to swap the values of two variables in the program. With the traditional approach, this can be done by using a temporary variable for swapping the values of two variables.

Example

```
>>>temp = x
>>> x = y
>>> y = temp
```

In order to swap the values of variables `x` and `y`, we need a temporary variable `temp`. However, this problem can be solved much more conveniently with tuple assignment.

```
>>>x = 3
>>>y = 4
>>>x , y = y , x          # Using tuple assignment
>>>print x
4              # Output
>>>print y
3              # Output
```

Hence, with the use of tuple assignment approach, there is no need to use any temporary variable to swap values of two variables. All it takes is one simple statement.

FEATURES OF TUPLE-TUPLE AS RETURN VALUE

Tuples can also be returned by the function as return values. Generally, the function returns only one value but by returning tuple, a function can return more than one value.

For example, if we want to compute a division with two integers and want to know the quotient and the remainder, both the quotient and the remainder can be computed at the same time. Two values will be returned, i.e., quotient and remainder, by using the tuple as the return value of the function.

Example

```
>>>defdiv_mod(a,b):                # defining function
... quotient = a/b
... remainder = a%b
... return quotient,remainder # function returning two values

# function calling
>>>x = 10
>>>y = 3
>>>t = div_mod(x,y)
>>>print t
(3, 1)                  # Output
>>>type(t)
<type 'tuple'>                  # Output
```

FEATURES OF TUPLE-TUPLE AS RETURN VALUE

In the given example, we have defined the function `div_mod`, which calculates the quotient and remainder. It returns two values, the quotient and the remainder respectively. Now, at the time of calling the function, a tuple needs to store the values returned by the function. Hence, we have taken a variable that calls the function `div_mod` and stores the values `(3,1)`, which, in our example, are quotient and remainder respectively.

When we tried to see the type of the variable `t`, it was tuple.

We can also use the tuple assignment approach in order to print the quotient and the remainder separately.

```
>>>quot, rem = div_mod(10,3)
>>> print quot
3            # Output
>>> print rem
1            # Output
```

Here, we have taken two variables at the left side which are `quot` and `rem`. Now, when the function `div_mod` returns the values of quotient and remainder, the values will be stored in `quot` and `rem` respectively.

BASIC TUPLE OPERATIONS

1. Concatenation The concatenation operator works in tuples in the same way as it does in lists. This operator concatenates two tuples. This is done by the + operator in Python.

Example

```
>>>t1 = (1,2,3,4)
>>>t2 = (5,6,7,8)
>>>t3 = t1 + t2
>>>print t3
(1, 2, 3, 4, 5, 6, 7, 8)                    # Output
```

In this example, there are two tuples, `t1` and `t2`. Tuples `t1` and `t2` are concatenated using + operator between them and the resulting tuple is stored in the variable `t3`. Now, when we print `t3`, it gives the concatenation of `t1` and `t2`.

BASIC TUPLE OPERATIONS

2. Repetition The repetition operator works as its name suggests; it repeats the tuples a given number of times. Repetition is performed by the* operator in Python.

Example

```
>>>tuple = ('ok',)
>>>tuple * 5
('ok', 'ok', 'ok', 'ok', 'ok')        # Output
>>>('Hello',) * 3
('Hello', 'Hello', 'Hello')           # Output
```

Note that, the tuple `('ok',)` was repeated 5 times and the tuple `('Hello',)` was repeated 3 times.

BASIC TUPLE OPERATIONS

3. `in` Operator The `in` operator also works on tuples. It tells user that the given element exists in the tuple or not. It gives a Boolean output, that is, TRUE or FALSE. If the given input exists in the tuple, it gives the TRUE as output, otherwise FALSE.

Example 1

```
>>>tuple = (10,20,30,40)
>>>20 in tuple
True                # Output
>>>50 in tuple
False               # Output
```

Example 2

```
>>>tuple = ('anil', 'rahul', 'rohan')
>>> 'james' in tuple
False               # Output
>>> 'rohan' in tuple
True          # Output
```

BASIC TUPLE OPERATIONS

4. Iteration Iteration can be done in tuples using `for` loop. It helps in traversing the tuple.

Example

```
>>>tuple = (1,2,3,4,5,6)
>>>for x in tuple:
... print x
```

Output:

```
1
2
3
4
5
6
```

BUILT IN TUPLE FUNCTIONS

S.No.	Function	Description
1.	`cmp(tuple1, tuple2)`	It compares the items of two tuples.
2.	`len(tuple)`	It returns the length of a tuple.
3.	`zip(tuple1, tuple2)`	It 'zips' elements from two tuples into a list of tuples.
4.	`max(tuple)`	It returns the largest value among the elements in a tuple.
5.	`min(tuple)`	It returns the smallest value among the elements in a tuple
6.	`tuple(seq)`	It converts a list into a tuple.

Example

```
>>> tuple1 = ('physics','chemistry','mathematics')
>>> tuple2 = (10,20,30,40,50)
>>>zip(tuple1,tuple2)
[('physics', 10), ('chemistry', 20), ('mathematics', 30)]
>>>max(tuple1)
'physics'          # Output
>>>max(tuple2)
50          # Output
>>>min(tuple1)
'chemistry'                # Output
>>>min(tuple2)
10          # Output
```

In this example, **physics** is max element of tuple1 because if we compare the first letter of all the words in the tuple then p is greater than c and m. Hence, the comparison stops here and physics is declared as the max element of the tuple. Similarly, **chemistry** is the min element of this tuple.

OPERATIONS OF DICTIONARIES

DICTIONARY IN PYTHON PROGRAMMING

The Python dictionary is an unordered collection of items or elements. All other compound data types in Python have only values as their elements or items whereas the dictionary has a key: value pair. Each value is associated with a key. In the list and the tuple, there are indices that are only of integer type but in dictionary, we have keys and they can be of any type.

Dictionary is said to be a mapping between some set of keys and values. Each key is associated to a value. The mapping of a key and value is called as a key-value pair and together they are called one item or element.

A key and its value are separated by a colon (:) between them. The items or elements in a dictionary are separated by commas and all the elements must be enclosed in curly braces. A pair of curly braces with no values in between is known as an empty dictionary.

The values in a dictionary can be duplicated, but the keys in the dictionary are unique.

CREATING A DICTIONARY

Creating a dictionary is simple in Python. The values in a dictionary can be of any data type, but the keys must be of immutable data types (such as string, number or tuple).

Example

Empty Dictionary

```
>>> dict1 = {}
>>>print dict1
{}          # Output
```

Dictionary with integer keys

```
>>> dict1 = {1:'red',2:'yellow',3:'green'}
>>>print dict1
{1: 'red', 2: 'yellow', 3: 'green'}          # Output
```

Dictionary with mixed keys

```
>>> dict1 = {'name' : 'jinnie', 3:['Hello',2,3]}
>>>print dict1
{3: ['Hello', 2, 3], 'name': 'jinnie'}              # Output
```

CREATING A DICTIONARY

In the above examples, we have seen many types of ways for creating a dictionary in Python programming language. One thing to be noticed in initialization of dictionary is that the values of keys can be given in any order but on printing the dictionary, it prints the sorted order of keys. This is because the dictionary has an internal mechanism to sort the keys and then print them.

Example

```
>>> d1 = dict({1:'red', 2:'yellow', 3:'green'})
>>> d2 = dict([(1,'red'), (2,'yellow'),(3,'green')])
>>> d3 = dict(one=1, two=2, three=3)
>>> print d3
{'three': 3, 'two': 2, 'one': 1}
```

In the above example, three dictionaries `d1, d2 and d3` are initialized using the built-in `dict` function in the Python programming language.

ACCESSING ELEMENT IN DICTIONARY

In order to access the elements from a dictionary, we can use the value of the key enclosed in square brackets. Python also provides a `get()` method that is used with the key in order to access the value. There is a difference in both the accessing methods. When the key is not found in the dictionary, it returns `none` instead of `KeyError`.

```
>>> dict1 = {'name' : 'John', 'age' : 27}
>>> dict1['name']
'John'                  # Output
>>> print dict1['name']
John          # Output
>>> print dict1['age']
27            # Output
>>> dict1.get('name')
'John'                  # Output
>>> print dict1.get('name')
John          # Output
>>> dict1.get('age')
27            # Output
```

UPDATING DICTIONARY

Dictionaries in Python are mutable. Unlike those in tuple and string, the values in a dictionary can be changed, added or deleted. If the key is present in the dictionary, then the associated value with that key is updated or changed; otherwise a new key: value pair is added.

Example

```
>>> dict1 = {'name' : 'John', 'age' : 27}
>>> dict1['age'] = 30    # updating a value
>>> print dict
{'age': 30, 'name': 'John'}              # Output
>>> dict1['address'] = 'Alaska'          # adding a key: value
>>>print dict1
{'age': 30, 'name': 'John', 'address': 'Alaska'}          # Output
```

Note that we tried to reassign the value `'30'` to the key `'age'`, Python interpreter first searches the key in the dictionary. In our example, the key `'age'` exists. Hence, the value of `'age'` is updated to `30`. However, in the next statement, it does not find the key `'address'`; hence, the key: value `'address': 'Alaska'` is added to the dictionary.

DELETING ELEMENTS FROM DICTIONARY

The items or elements from a dictionary can be removed or deleted by using `pop()` method. `pop()` method removes that item from the dictionary for which the key is provided. It also returns the value of the item.

Furthermore, there is a `popitem()` method in Python. `popitem()` method is used to remove or delete and return an arbitrary item from the dictionary.

The `clear()` method removes all the items or elements from a dictionary at once. When this operation is performed, the dictionary becomes an empty dictionary.

Python also provides a `del` keyword, which deletes the dictionary itself. When this operation is performed, the dictionary is deleted from the memory and it ceases to exist.

DELETING ELEMENTS FROM DICTIONARY

Example

```
>>>dict_cubes = {1:1, 2:8, 3:9, 4:64, 5:125, 6:216}

>>>dict_cubes.pop(3)              # remove a particular item
9           # Output
>>>dict_cubes
{1: 1, 2: 8, 4: 64, 5: 125, 6: 216}          # Output

>>>dict_cubes.popitem()          # remove an arbitrary item
(1, 1)              # Output
>>>dict_cubes.popitem()
(2, 8)              # Output
>>>dict_cubes
{4: 64, 5: 125, 6: 216}          # Output

>>>deldict_cubes[6]              # delete a particular item
>>>dict_cubes
{4: 64, 5: 125}           # Output

>>>dict_cubes.clear()            # remove all items
>>>dict_cubes
{}          # Output
```

OPERATIONS IN DICTIONARY

1. Traversing We have learnt about traversing strings, lists and tuples in the previous sections. Now, we will learn traversing in a dictionary. Traversing in dictionary is done on the basis of keys. For this, `for` loop is used, which iterates over the keys in the dictionary and prints the corresponding values using keys.

Example

We will define a function `print_dict`. Whenever a dictionary is passed as an argument to this function, it will print the keys and values of the dictionary.

```
>>>defprint_dict(d):
... for c in d:
...         print c,d[c]

>>> dict1 = {1:'a',2:'b',3:'c',4:'d',5:'e',6:'f',7:'g',8:'h'}
>>> print_dict(dict1)
```

Output:

```
1 a
2 b
3 c
4 d
5 e
6 f
7 g
8 h
```

OPERATIONS IN DICTIONARY

2. Membership Using the membership operator (`in` and `not in`), we can test whether a key is in the dictionary or not. We have seen the `in` operator earlier as well in the list and the tuple. It takes an input key and finds the key in the dictionary. If the key is found, then it returns True, otherwise, False.

Example

```
>>>cubes = {1:1, 2:8, 3:27, 4:64, 5:125, 6:216}
>>>3 in cubes
True        # Output
>>>7 not in cubes
True        # Output
>>>10 in cubes
False               # Output
```

BUILD IN DICTIONARY METHODS

Sr.No.	Function	Description
1.	`all(dict)`	It is a Boolean type function, which returns True if all keys of dictionary are true (or the dictionary is empty).
2.	`any(dict)`	It is also a Boolean type function, which returns True if any key of the dictionary is true. It returns false if the dictionary is empty.
3.	`len(dict)`	It returns the number of items (length) in the dictionary.
4.	`cmp(dict1,dict2)`	It compares the items of two dictionaries.
5.	`sorted(dict)`	It returns the sorted list of keys.
6.	`str(dict)`	It produces a printable string representation of the dictionary.
7.	`dict.clear()`	It deletes all the items in a dictionary at once.
8.	`dict.copy()`	It returns a copy of the dictionary.
9.	`dict.fromkeys()`	It creates a new dictionary with keys from sequence and values set to value.
10.	`dict.get(key, default=None)`	For `key` key, returns value or default if key not in dictionary.
11.	`dict.has_key(key)`	It finds the key in dictionary; returns True if found and false otherwise.
12.	`dict.items()`	It returns a list of entire key: value pair of dictionary.
13.	`dict.keys()`	It returns the list of all the keys in dictionary.
14.	`dict.setdefault (key, default=None)`	Similar to `get()`, but will set `dict[key]=default` if `key` is not already in dict.
15.	`dict.update(dict2)`	It adds the items from `dict2` to `dict`.

ThankYou