TUPLES

Agenda

- Introduction
- Creating a tuple
- Accessing Elements in a tuple
- Packing and unpacking
- Comparing tuples
- Using tuples as keys in dictionary
- Deleting tuples
- Python Tuple methods
- Built-in functions with Tuple
- Advantages of tuple over list

Introduction

- A tuple is just like a list of a sequence of immutable python objects.
- The difference between list and tuple is that list are declared in square brackets and can be changed while tuple is declared in parentheses and cannot be changed.
- However, you can take portions of existing tuples to make new tuples.

Creating a Tuple

- A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma. The parentheses are optional but is a good practice to write it.
- A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).

```
>>> my_tuple = ()

>>> print(my_tuple)
()
>>> int_tuple=(1,2,3)
>>> print(int_tuple)
(1, 2, 3)
>>> mixed_tuple=(1,"Hello",3.4)
>>> print(mixed_tuple)
(1, 'Hello', 3.4)
>>> nested_tuple=("mouse",[8,4,6],(1,2,3))
>>> print(nested_tuple)
('mouse', [8, 4, 6], (1, 2, 3))
>>>
```

```
>>> a= 1,2,3
>>> a
(1, 2, 3)
>>>
```

Accessing Elements in a Tuple

- **<u>Forward and backward Indexing:</u>**We can use the index operator [] to access an item in a tuple where the forward index starts from 0 and backward index starts from -1 ,-2 and so on

```
>>> a=(10,20,30,[90,100],89,78)
>>> a[0]
10
>>> a[1]
20
>>> a[-1]
78
>>> a[-2]
89
>>> a[3][0]
90
>>> a[3][1]
100
>>>
```

Accessing Elements in a Tuple

- **Slicing:**We can access a range of items in a tuple by using the slicing operator - colon ":"

```
>>> my_tuple = ('A','P','S','S','D','C')
>>> print(my_tuple[1:4])
('P', 'S', 'S')
>>> print(my_tuple[:-3])
('A', 'P', 'S')
>>> print(my_tuple[3:])
('S', 'D', 'C')
>>> print(my_tuple[:])
('A', 'P', 'S', 'S', 'D', 'C')
>>>
```

Continue….

- Tuples are immutable; you can't change which variables they contain after construction. However, you can concatenate or slice them to form new tuples:

Python 3.4.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 3
2 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=(1,2,3)
>>> a[0]
1
>>> a[0]=200
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    a[0]=200
TypeError: 'tuple' object does not support item assignment
>>> a[2]
3
>>>
```

Continue…

Python 3.4.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600
32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=(1,2,3)
>>> a[0]
1
>>> a[1]
2
>>> a[2]
3
>>> a[3]=50
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    a[3]=50
TypeError: 'tuple' object does not support item assignment
>>> b=a+(4,5,6)
>>> b
(1, 2, 3, 4, 5, 6)
>>> c=b[1:]
>>> c
(2, 3, 4, 5, 6)
>>>
```

Continue….

Python has tuple assignment feature which enables you to assign more than one variable at a time.

```
>>> name="Ram"
>>> age=25
>>> location="Vijayawada"
>>> ram=(name,age,location)
>>> ram
('Ram', 25, 'Vijayawada')
>>>
```

```
>>> ('name','surname','age')=robert
SyntaxError: can't assign to literal
>>> robert=('name','surname','age')
>>> robert
('name', 'surname', 'age')
>>>
```

Packing and Unpacking

- In packing, we place value into a new tuple while in unpacking we extract those values back into variables

Python 3.4.4 Shell

File Edit Shell Debug Options Window Help

```
>>> x = ("Apssdc", 20, "Education")   # tuple packing
>>> (company, emp, profile) = x   # tuple unpacking
>>> print(company)
Apssdc
>>> print (emp)
20
>>> print (profile)
Education
>>>
```

Ln: 43 Col: 4

Comparing Tuples

- A comparison operator in Python can work with tuples.
- The comparison starts with a first element of each tuple. If they do not compare to =, < or > then it proceed to the second element and so on.
- It starts with comparing the first element from each of the tuples

Example

```
>>> a=(5,6)
>>> b=(1,4)
>>> if(a>b):
        print("a is bigger")
else:
        print("b is bigger")


a is bigger
>>> a=(5,6)
>>> b=(8,4)
>>> if(a>b):
        print("a is bigger")
else:
        print("b is bigger")


b is bigger
>>> a=(5,6)
>>> b=(5,4)
>>> if(a>b):
        print("a is bigger")
else:
        print("b is bigger")


a is bigger
>>>
```

Using Tuples as Keys in Dictionaries

Since tuples are hashable, and list is not, we must use tuple as the key if we need to create a composite key to use in a dictionary.

Example

- suppose I have quantities of fruits of different colors, e.g., 24 blue bananas, 12 green apples, 0 blue strawberries and so on. I'd like to organize them in a data structure in Python that allows for easy selection and sorting. My idea was to put them into a dictionary with tuples as keys, e.g.,

```
>>> fruits={('banana','blue'):24,('apple','green'):12,('strawberry','blue' ):0,}
>>> fruits
{('strawberry', 'blue'): 0, ('banana', 'blue'): 24, ('apple', 'green'): 12}
>>>
```

Tuples and Dictionary

- Dictionary can return the list of tuples by calling items, where each tuple is a key value pair.

```
>>> a={'x':100,'y':200}
>>> b=a.items()
>>> print(b)
dict_items([('y', 200), ('x', 100)])
```

Deleting Tuples

- Tuples are immutable and cannot be deleted, but deleting tuple entirely is possible by using the keyword "del."

```
>>> x=('a','b','c','d','e')
>>> x[0]
'a'
>>> del x[1]
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    del x[1]
TypeError: 'tuple' object doesn't support item deletion
```

```
>>> x=('a','b','c','d','e')
>>> x
('a', 'b', 'c', 'd', 'e')
>>> del x
>>> x
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    x
NameError: name 'x' is not defined
```

Python Tuple Methods

- Methods that add items or remove items are not available with tuple. Only the following two methods are available.

Method	Description
count(x)	Return the number of items that is equal to x
index(x)	Return index of first item that is equal to x

```
>>> my_tuple = ('a','p','p','l','e',)
>>> print(my_tuple.count('p'))
2
>>> print(my_tuple.index('l'))
3
>>>
```

Tuple Membership Test

We can test if an item exists in a tuple or not, using the keyword “in”

Python 3.4.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> my_tuple = ('a','p','p','l','e',)
>>> print('a' in my_tuple)
True
>>> print('b' in my_tuple)
False
>>>
```

Built in Functions with Tuples

- To perform different task, tuple allows you to use many built-in functions like all (), any (), enumerate (), max (), min (), sorted (), len (), tuple (), etc

Built in Functions with Tuples

Function	Description
all()	Return `True` if all elements of the tuple are true (or if the tuple is empty).
any()	Return `True` if any element of the tuple is true. If the tuple is empty, return `False.`
enumerate()	Return an enumerate object. It contains the index and value of all the items of tuple as pairs.
len()	Return the length (the number of items) in the tuple.
max()	Return the largest item in the tuple.
min()	Return the smallest item in the tuple
sorted()	Take elements in the tuple and return a new sorted list (does not sort the tuple itself).
sum()	Retrun the sum of all elements in the tuple.
tuple()	Convert an iterable (list, string, set, dictionary) to a tuple.

Advantages of tuple over list

- Iterating through tuple is faster than with list, since tuples are immutable.
- Tuples that consist of immutable elements can be used as key for dictionary, which is not possible with list
- If you have data that is immutable, implementing it as tuple will guarantee that it remains write-protected