A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.

Advantages of Tuple over List

Since, tuples are quite similar to lists, both of them are used in similar situations as well.  
However, there are certain advantages of implementing a tuple over a list.

Below listed are some of the main advantages:

• We generally use tuple for heterogeneous (different) data types and list for homogeneous (similar) datatypes.  
• Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.  
• Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.  
• If you have data that doesn’t change, implementing it as tuple will guarantee that it remains write-protected.

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.).

# empty tuple  
n = ()

# tuple having integers  
n = (1, 2, 3)

# tuple with mixed datatypes  
n = (1, “Hello”, 3.4)

# nested tuple  
n = (“mouse”, [8, 4, 6], (1, 2, 3))

Creating a tuple with one element is a bit tricky.

# only parentheses is not enough

n = (“hello”)  
print(type(n))  
# Output: <class ‘str’>

# need a comma at the end

n = (“hello”,)  
print(type(n))  
# Output: <class ‘tuple’>

# parentheses is optional

n = “hello”,  
print(type(n))  
# Output: <class ‘tuple’>

Creating tuples from existing sequences

We can use the built-in type object (tuple()) to create tuple from sequences using the syntax given below.

t=tuple(<sequence>)  
where <sequence> can be any kind of sequence object including string , list and tuples.

Python creates the individual elements of the tuple from the individual elements of passed sequence. If we pass in another tuple , the tuple function make a copy.

Example:  
t1=tuple(‘computer’)  
print(t)

Output:  
(‘c’, ‘o’, ‘m’, ‘p’, ‘u’, ‘t’, ‘e’, ‘r’)  
>>>

Example:  
t2=tuple(‘hello’)  
print(t2)  
Output:  
(‘h’, ‘e’, ‘l’, ‘l’, ‘o’)  
>>>

We can also create a tuple by taking input from the user.  
Example:  
n=input(“Enter tuple elements “)  
t1=tuple(n)  
print(t1)  
Output:

Enter tuple elements Computer  
(‘C’, ‘o’, ‘m’, ‘p’, ‘u’, ‘t’, ‘e’, ‘r’)  
>>>

The most commonly use method to input a tuple is eval(input()) as given below:  
Example:  
n=eval(input(“Enter tuple elements “))  
print(n)  
t1=tuple(n)  
print(t1)

Output:  
Enter tuple elements (1,2,3,’hello’,10)  
(1, 2, 3, ‘hello’, 10)  
(1, 2, 3, ‘hello’, 10)  
>>>

## Accessing Elements in a Tuple

There are various ways in which we can access the elements of a tuple.

## Indexing

* We can use the index operator [] to access an item in a tuple where the index starts from 0.
* So, a tuple having 6 elements will have index from 0 to 5. Trying to access an element other that (6, 7,…) will raise an IndexError.
* The index must be an integer, so we cannot use float or other types. This will result into TypeError.
* Likewise, nested tuple are accessed using nested indexing, as shown in the example below.

c=(‘c’,’o’,’m’,’p’,’u’,’t’,’e’,’r’)  
Positive index

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| c | o | m | p | u | t | e | r |

Example:

n=(‘c’,’o’,’m’,’p’,’u’,’t’,’e’,’r’)  
print(n)  
print(n[1])  
print(n[3])  
print(n[4])  
print(n[5])  
print(n[7])

Output:

Output:

(‘c’, ‘o’, ‘m’, ‘p’, ‘u’, ‘t’, ‘e’, ‘r’)  
o  
p  
u  
t  
r

n=('c','o','m','p','u','t','e','r')

print(n)

print(n[1])

print(n[3])

print(n[4])

print(n[5])

print(n[7])

Output:

(‘c’, ‘o’, ‘m’, ‘p’, ‘u’, ‘t’, ‘e’, ‘r’)  
o  
p  
u  
t  
r

n=('c','a','t','a','l','y','s','t')

print(n)

print(n[1])

print(n[3])

print(n[2])

print(n[6])

print(n[4])

Output:

(‘c’, ‘a’, ‘t’, ‘a’, ‘l’, ‘y’, ‘s’, ‘t’)  
a  
a  
t  
s  
l

n=(10,20,30,40,50,60,70)

print(n)

print(n[1])

print(n[3])

print(n[2])

print(n[6])

print(n[4])

Output:

(10, 20, 30, 40, 50, 60, 70)  
20  
40  
30  
70  
50

Negative Indexing

Python allows negative indexing for its sequences.  
The index of -1 refers to the last item, -2 to the second last item and so on.

Slicing

We can access a range of items in a tuple by using the slicing operator – colon “:”.

## Changing a Tuple

Unlike lists, tuples are immutable.  
This means that elements of a tuple cannot be changed once it has been assigned. But, if the element is itself a mutable data type like list, its nested items can be changed.

**We can assign a tuple to different values (reassignment).**  
n=(0,1,2,3,4,5,6,7,8,9,10)  
print(n)  
output:  
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

n[1]=10  
**#error will get generated as tuple elements cannot be changed**  
print(n)

**#but list within a tuple can be changed**  
n = (4, 2, 3, [6, 5])  
print(n)  
output:  
(4, 2, 3, [6, 5])

n[3][0]=10  
print(n)  
Output:  
(4, 2, 3, [10, 5])  
n[3][1]=100  
print(n)  
Output:  
(4, 2, 3, [10, 100])