PES UNIVERSITY

Department of Computer Science and Engineering

PES University, Bangalore, India

Python For Computational Problem Solving (UE22CS151A)

Mr. Prakash C O
Asst. Professor,
Dept. of CSE, PESU
coprakasha@pes.edu

Tuples - Immutable Groups of Objects

A tuple is a collection which is ordered, unchangeable (immutable) and heterogeneous. In Python tuples are written with round brackets (parenthesis).

fruitstuple = ("apple", "banana", "cherry", "orange")

Tuples are identical to lists in all respects, except for the following properties:

- Tuples are defined by enclosing the elements in parentheses (()) instead of square brackets ([]).
- Tuples are immutable—they can't be modified once they've been created.

A tuple is a data structure with the following attributes.

- A tuple has zero or more elements.
- The element of the tuple can be of any type. There is no restriction on the type of the element.
- A tuple has elements which could be of the same type (homogeneous) or of different types(heterogeneous)
- Each element of a tuple can be referred to by a index or a subscript. An index is an integer
- Access to an element based on index or position takes the same time no matter where the element is in the tuple – random access.
- Tuples are immutable. Once created, we cannot change the number of elements – no append, no insert, no remove, no delete.
- Elements of the tuple cannot be assigned.
- A tuple cannot grow and cannot shrink. Size of the tuple can be found using the function len.
- Elements in the tuple can repeat
- Tuple is a sequence
- Tuple is also iterable - is eager and not lazy.
- Tuples can be nested. We can have tuple of tuples.
- Assignment of one tuple to another causes both to refer to the same tuple.
- Tuples can be sliced. This creates a new tuple
- You may walk through this program to understand the creation and the operations on a tuple.
- If the element of a tuple is a list, then the list can be modified by adding elements or removing them.
- We cannot replace the element of the tuple by assignment even if it is a list.

Why use a tuple instead of a list?

- Program execution is faster when manipulating a tuple than it is for the equivalent list. (This is probably not going to be noticeable when the list or tuple is small.)
- Sometimes you don’t want data to be modified. If the values in the collection are meant to remain constant for the life of the program, using a tuple instead of a list guards against accidental modification.
- There is another Python data type that you will encounter shortly called a dictionary, which requires as one of its components a value that is of an immutable type. A tuple can be used for this purpose, whereas a list can’t be.

Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Creation of tuple:

- **Empty tuple**
  - `t1 = ()`
  - `t2 = tuple()`
- **Two element tuple**
  - `t3 = (1, 2)`
- **One element tuple**
  - `t4 = (10) # NO`
  - `t5 = (20,) # Yes`

There is an ambiguity when we use parentheses around a single expression. Should we consider this as an expression or a tuple? The language considers this as an expression. To make a tuple of single element, we require an extra comma.

In [1]:
```
# Creation of tuple
t1 = ()
print(t1, type(t1))

t2 = tuple()
print(t2, type(t2))
```

```
() <class 'tuple'>
() <class 'tuple'>
```

In [3]:
```
t3 = (1, 2)
print(t3, type(t3))

t4 = (10) # its not a tuple, its an expression
```

```
print(t4, type(t4))

t5 = (20,) # Yes, single element tuple
print(t5, type(t5))
```

```
(1, 2) <class 'tuple'>
10 <class 'int'>
(20,) <class 'tuple'>
```

Tuple characteristics

In [4]:
```
# tuple
    #   is a sequence
    #   like a list
    #   indexed by int; leftmost element has an index 0
    #   select the element using []
a = (11, 33, 22, 44, 55)
b =  11, 33, 22, 44, 55      # Parentheses are optional.
print('a =',a)
print('b =',b)
print(a[2])     # 22
print(a[:-2])   # leaving last 2 elements
print(a[-2:])   # taking last 2 elements
```

```
a = (11, 33, 22, 44, 55)
b = (11, 33, 22, 44, 55)
22
(11, 33, 22)
(44, 55)
```

In [3]:
```
# tuple
    #   is immutable
    #   once created, cannot be changed
    #   length of the tuple cannot change
a = (11, 33, 22, 44, 55)
#a[2] = 222    # Error - 'tuple' object does not support item assignment
#a.append(66) # Error - 'tuple' object has no attribute 'append'
```

In [8]:
```
a=2
b=5.5
t1=(a,b)
print('t1 =',t1)
print('id(t1) =',id(t1))
a=3
t2=(a,b)
print('t2 =',t2)
print('id(t2) =',id(t2))
```

```
t1 = (2, 5.5)
id(t1) = 1705898215104
t2 = (3, 5.5)
id(t2) = 1705898216000
```

In [4]:
```
a=2
b=5.5
t1=(a,b)
t2=t1
print(id(t1))
print(id(t2))
```

```
2131882044096
2131882044096
```

In [2]:
```
a = (11, 22, 33, 44, 55)
b = (66, 77)
# we can create new tuples from existing tuples
```

```
a = a + b + (111, 222)
print(a)        #(11, 22, 33, 44, 55, 66, 77, 111, 222)
print(b)
```

```
(11, 22, 33, 44, 55, 66, 77, 111, 222)
(66, 77)
```

In [9]:
```
# tuple
#       is heterogeneous
b = ([12, 23], {1:'Good',2:'Bad'}, "56" )
print(b, len(b))

b[0].append(67)    #ok
print(b, len(b))
# b[0] = [78, 89]       # no ; changing tuple element is not allowed
# del b[0]              # no ; deleting tuple element is not allowed
# b[0] = b[0] + [100]   # no ; assignment forbidden
```

```
([12, 23], {1: 'Good', 2: 'Bad'}, '56') 3
([12, 23, 67], {1: 'Good', 2: 'Bad'}, '56') 3
```

In [13]:
```
# tuple
#       is iterable
t = (11, 22, 33, 44)
for i in t:
    print(i, end=' ')
```

```
11 22 33 44
```

In [27]:
```
c = [11, 22, 33, 44]
for i in c :
    print("one")
```

```
one
one
one
one
```

In [9]:
```
c = [11, 22, 33, 44]
print('c =',c)
d=[c]
print('d =',d)
for i in d:
    print('two')
```

```
c = [11, 22, 33, 44]
d = [[11, 22, 33, 44]]
two
```

In [12]:
```
c = [11, 22, 33, 44]
d=(c)         # (c) -> expression having list named c
print('d =',d)
#print('id(c) =',id(c))
#print('id(d) =',id(d))
e=tuple(c)
print('e =',e)
for i in (c) : # (c) -> not a tuple, (c) or c both mean the same
    print("three")
```

```
d = [11, 22, 33, 44]
id(c) = 2131882051968
id(d) = 2131882051968
e = (11, 22, 33, 44)
three
three
three
three
```

In [29]:
```
c = [11, 22, 33, 44]
d=(c,)     # tuple of one element requires an extra comma
print(d)
for i in (c,) : # a tuple
    print("four")
```
```
([11, 22, 33, 44],)
four
```

In [12]:
```
t1=(1,2,3)
print(t1)
t2=(t1)
print(t2)
t3=(t1,)  # nested tuple
print(t3)

t4=(1,2,3)
t5=[t4]
print(t5)
```
```
(1, 2, 3)
(1, 2, 3)
((1, 2, 3),)
[(1, 2, 3)]
```

In [7]:
```
print( (3, 4) * 2)
print( (3 + 4) * 2)
print( (3 + 4,) * 2)
# tuple of one element requires an extra comma
d = ()
print(d, type(d))
```
```
(3, 4, 3, 4)
14
(7, 7)
() <class 'tuple'>
```

In [ ]:
```
print( (3, 4) * 2) # (3, 4, 3, 4)
print( (3 + 4) * 2) # 14
print( (3 + 4,) * 2) # (7, 7)
# tuple of one element requires an extra comma
d = ()
print(d, type(d))
```

In [30]:
```
e = (11, 33, 11, 11, 44, 33)
print(e.count(11)) # 3
print(e.count(33)) # 2
print(e.count(55)) # 0
print(e.index(44)) # 4
print(e.index(11)) # 0
# print(e.index(55)) # value error - value not in tuple
```
```
3
2
0
4
0
```

In [32]:
```
# In this example, we will consider a few uses of tuples.
a = 1, 2, 3  # Parentheses are optional.
# The above statement creates a tuple of 3 elements.
print(a)
```
```
(1, 2, 3)
```

In [8]:
```
a = 1, 2, 3
print(a)
```

```
x, y, z= a
print(x, y, z)
# This will cause the elements of the tuple to be assigned to x, y and z.
```

```
(1, 2, 3)
1 2 3
```

In [13]:
```
# We may also use unnamed tuples while assigning to number of variables.
a, b = 11, 22
# The corresponding elements are assigned thereby a becomes 11 and b becomes 22.

print('a =',a,'b =',b)
(a, b) = (b, a) # swaps two variables
#a, b = b, a # swaps two variables
print('a =',a,'b =',b)
```

```
a = 11 b = 22
a = 22 b = 11
```

In [17]:
```
a = 1, 2, 3
print(a, type(a))

x, y, z = a
print(x, y, z)

#x, y = a    # error; # of variables on the left should match the # of elem in the tuple
            # ValueError: too many values to unpack (expected 2)
```

```
(1, 2, 3) <class 'tuple'>
1 2 3
```

In [37]:
```
# use of unnamed tuple
a, b = 11, 22       # equivalent to (a, b) = 11, 22
print("a : ", a, " b : ", b)
# in case of assignment, the right hand side is completely evaluated before
# assignment
(a, b) = (b, a) # swaps two variables  # (a, b) = (22, 11)
print("a : ", a, " b : ", b)
```

```
a :  11  b :  22
a :  22  b :  11
```

In [2]:
```
a=5; b=5.5; c='Hi'; d=257
t=(a,b,c,d)
print(t)

a=6; d=290
print(t)
```

```
(5, 5.5, 'Hi', 257)
(5, 5.5, 'Hi', 257)
```

In [3]:
```
val1=1,23,000
print(val1)
val2=1,23,.000
print(val2)
val3=1+2,.024
print(val3)
val3=3*(1+2,24)
print(val3)
```

```
(1, 23, 0)
(1, 23, 0.0)
(3, 0.024)
(3, 24, 3, 24, 3, 24)
```

In [4]:
```
if (()):          # empty tuple represents Boolean value False
```

```
    print('Hello')
else:
    print('Hi')
```

```
Hi
```

In [5]:
```
t = 2,
print(t)
```

```
(2,)
```

In [6]:
```
x1, x2, x3 = 4, 5, 6
print(x1, x2, x3)
```

```
4 5 6
```

Underscore for Ignoring the values

The underscore is also used for ignoring the specific values. If you don’t need the specific values or the values are not used, just assign the values to underscore.

When unpacking, the number of variables on the left must match the number of values in the tuple.

In [32]:
```
t = (1,2,3,4)
print('t =',t)
a,b,c,d = t
print(a,b,c,d)
```

```
t = (1, 2, 3, 4)
1 4 3 4
```

In [36]:
```
t = (1,2,3,4)
print('t =',t)
a,_,c,_ = t    #The underscore is also used for ignoring the specific values
print(a,_,c,_) #Python stores the last expression value to the special variable called '
```

```
t = (1, 2, 3, 4)
1 4 3 4
```

In [35]:
```
# Ignore the index
for _ in range(10):
    pass
```

In [34]:
```
for _ in range(10):
    print(_,end=' ')
```

```
0 1 2 3 4 5 6 7 8 9
```

In [33]:
```
list_of_tuple = [(1,'Hi'),(2,'Hello'),(3,'How are you?')]
# Ignore a value of specific location
for _, val in list_of_tuple:
    print(val)
```

```
Hi
Hello
How are you?
```

In [1]:
```
t = (True, 0, False, 1)
print(sum(t))
```

```
2
```

Conclusion

We covered the basic properties of Python lists and tuples, and how to manipulate them.

One of the chief characteristics of a list is that it is ordered. The order of the elements in a list is an intrinsic property of that list and does not change, unless the list itself is modified. (The same is true of tuples, except of course they can’t be modified.)