Tuples

July 22, 2022

Python Data Structures - Tuples

Tuples

A tuple represents a sequence of any objects separated by commas and enclosed in parentheses( ). In some ways, a tuple is similar to a list in terms of indexing, nested objects, and repetition But a tuple is immutable, unlike lists which are mutable

Features of Python Tuple :

The tuple is an immutable data type i.e, Tuples are not modifiable.

Tuples are ordered sequence which means that all the elements have a defined order

The element of the tuple can access by index

A tuple can have any number of items and they may be of different types (Heterogenous)

A tuple may contain duplicate values

Tuple are iterable.

Creating a Tuple

[3]:
```
#Tuples in Python can be created by just placing the sequence inside the ( )␣
 ↪separated by commas (,).

# Empty tuple
emptyTuple = ()
print(emptyTuple,"-->",type(emptyTuple))

# Tuple having integers
integersTuple = (1, 2, 3)
print(integersTuple,"-->",type(integersTuple))

# tuple with mixed datatypes
mixedTuple = (1, "Hello", 3.4)
print(mixedTuple,"-->",type(mixedTuple ))

# nested tuple
Nestedtuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(Nestedtuple,"-->",type(Nestedtuple))
```

```
() --> <class 'tuple'>
(1, 2, 3) --> <class 'tuple'>
(1, 'Hello', 3.4) --> <class 'tuple'>
('mouse', [8, 4, 6], (1, 2, 3)) --> <class 'tuple'>
```

[4]:
```
#A tuple can also be created without using parentheses. This is known as tuple␣
↪packing.

my_tuple = 3, 4.6, "dog"
print(my_tuple,"-->",type(my_tuple))

# tuple unpacking is also possible
a, b, c = my_tuple

print(a)      # 3
print(b)      # 4.6
print(c)      # dog
```

```
(3, 4.6, 'dog') --> <class 'tuple'>
3
4.6
dog
```

[5]:
```
#Creating a tuple with one element is a bit tricky.

my_tuple = ("hello")
print(my_tuple,"-->",type(my_tuple))  # <class 'str'>

# Creating a tuple having one element
my_tuple = ("hello",)
print(my_tuple,"-->",type(my_tuple))  # <class 'tuple'>

# Parentheses is optional
my_tuple = "hello",
print(my_tuple,"-->",type(my_tuple))  # <class 'tuple'>
```

```
hello --> <class 'str'>
('hello',) --> <class 'tuple'>
('hello',) --> <class 'tuple'>
```

Accessing elements from the Tuple Using Indexing and Slicing

[6]:
```
#Indexing
a_tuple = (0, [1, 2, 3], (4, 5, 6), 7.0)

print('The first element:', a_tuple[0])
print('The last element:', a_tuple[-1])
print('The second element of the inner tuple:',a_tuple[2][1])
print('The data type of the second element:', type(a_tuple[1]))
```

```
The first element: 0
The last element: 7.0
The second element of the inner tuple: 5
The data type of the second element: <class 'list'>
```

[7]:
```
#Slicing
num_tuple = 2, 4, 5, 7, 8, 10
print(num_tuple[:3])
print(num_tuple[4:])
print(num_tuple[-3:])
print(num_tuple[2:5])
```

```
(2, 4, 5)
(8, 10)
(7, 8, 10)
(5, 7, 8)
```

Updating / Changing Tuples in Python

[47]:
```
#Adding a new element or deleting one is not really an option when dealing with␣
 ↪tuples in python,
#as they are immutable. Even the elements of the tuple cannot be updated
#until and unless the element is mutable for example a list.

# Changing tuple values
my_tuple = (4, 2, 3, [6, 5])

# TypeError: 'tuple' object does not support item assignment
#my_tuple[1] = 9

# However, item of mutable element can be changed
my_tuple[3][0] = 9    # Output: (4, 2, 3, [9, 5])
my_tuple[3].pop(1)
print(my_tuple)

# Tuples can be reassigned - Assigning tuple all over again
my_tuple = ('Have', 'a', 'great', 'day', [1, 2, 3])
print(my_tuple)
```

```
(4, 2, 3, [9])
('Have', 'a', 'great', 'day', [1, 2, 3])
```

Deleting a Tuple

[10]:
```
#You can delete a tuple as a whole,
#but deleting a specific value/element in a tuple is not possible

tempTuple = (1, 2, 3, 4, 5)
```

```
# tempTuple.pop() # throws error as object has no attribute pop
# del tempTuple[3] # throws error as tuple does not support object deletion

print(tempTuple) # OUTPUT: (1, 2, 3, 4, 5)

del tempTuple
print(tempTuple) # throws NameError: name 'tempTuple' is not defined
```

```
(1, 2, 3, 4, 5)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-689e943e8a01> in <module>
     10
     11 del tempTuple
---> 12 print(tempTuple) # throws NameError: name 'tempTuple' is not defined

NameError: name 'tempTuple' is not defined
```

Basic Python Tuple Operations

[11]:
```
#Just like '+' and '*' operations, tuple also respond to other sequential␣
 ↪operations like
#length, membership and for loop
#Both + and * operations result in a new tuple.

tempTuple = ('apple', 'mango')
result = tempTuple + (1, 2, 3) # concatenating two different tuples
print('concatenation of a tuple', result) # OUTPUT: concatenation of a tuple␣
 ↪('apple', 'mango', 1, 2, 3)

result = tempTuple * 3 # repeat a tuple 3 times
print('repetition of a tuple', result) # OUTPUT: repetition of a tuple␣
 ↪('apple', 'mango', 'apple', 'mango', 'apple', 'mango')

result = len(tempTuple) # length of a tuple
print('length of the tuple:', result) # OUTPUT: length of the tuple: 2

result = 'apple' in tempTuple # checking membership of an element in tuple
print('membership check', result) # OUTPUT: membership check True
```

```
concatenation of a tuple ('apple', 'mango', 1, 2, 3)
repetition of a tuple ('apple', 'mango', 'apple', 'mango', 'apple', 'mango')
length of the tuple: 2
membership check True
```

Iterating Through a Tuple

[13]:
```
# Using a for loop to iterate through a tuple
for name in ('John', 'Kate'):
    print("Hello", name)
```

```
Hello John
Hello Kate
```

[14]:
```
#The enumerate function returns a tuple containing a count for every iteration
friends = ('Steve', 'Rachel', 'Michael', 'Monica')
for index, friend in enumerate(friends):
    print(index,friend)
```

```
0 Steve
1 Rachel
2 Michael
3 Monica
```

Usage of Tuples

[23]:
```
#One of the most common use cases of tuples is with functions that return␣
 ↪multiple values.

import numpy as np

#A function that returns count and sum of a array passed
def count_sum(arr):
   count = len(arr)
   sum = arr.sum()
   return count, sum


arr = np.random.randint(10, size=8)
a = count_sum(arr)
print(a)
(8, 39)
print(type(a))
```

```
(8, 22)
<class 'tuple'>
```

[28]:
```
#Tuples are useful for sequence unpacking.
x, y = (7, 10);
print("Value of x is {}, the value of y is {}.".format(x, y))
```

```
Value of x is 7, the value of y is 10.
```

[30]:
```
#Tuple Element Swapping
x = 19
y = 91
print('Before swapping:')
```

```
print(f'x = {x}, y = {y}')
(x, y) = (y, x)
print('After swapping:')
print(f'x = {x}, y = {y}')
```

```
Before swapping:
x = 19, y = 91
After swapping:
x = 91, y = 19
```

Tuple Methods

[25]:
```
#Methods that add items or remove items are not available with tuple.
#Only the index(),count() methods are available

my_tuple = ('a', 'p', 'p', 'l', 'e',)

print(my_tuple.count('p'))  # Output: 2
print(my_tuple.index('l'))  # Output: 3
```

```
2
3
```

Tuple Functions

[26]:
```
a=(3,1,2,5,4,6,0)
print(max(a))
print(min(a))
print(sum(a))
print(any(a))
print(all(a))
print(sorted(a))
```

```
6
0
21
True
False
[0, 1, 2, 3, 4, 5, 6]
```

[27]:
```
#tuple( )
list1=[1,2,3]
print(tuple(list1))

emptyTuple=tuple()
print(emptyTuple)
```

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

Zipping Tuples

[31]:
```
#The zip() method takes multiple sequence objects
#and returns an iterable object by matching their elements.

first_names = ('Simon', 'Sarah', 'Mehdi', 'Fatime')
last_names = ('Sinek', 'Smith', 'Lotfinejad', 'Lopes')
ages = (49, 55, 39, 33)
zipped = zip(first_names, last_names,ages)
print(zipped)
```

```
<zip object at 0x000002DE84B35500>
```

[32]:
```
# To consume the iterator object, we need to convert it to either a list or a␣
 ↪tuple
customers = tuple(zipped)
print(customers)
```

```
(('Simon', 'Sinek', 49), ('Sarah', 'Smith', 55), ('Mehdi', 'Lotfinejad', 39),
('Fatime', 'Lopes', 33))
```

[33]:
```
#Unpacking Tuples
first_name, last_name, age = customers[2]
print(first_name, last_name, ',', age, 'years old')
```

```
Mehdi Lotfinejad , 39 years old
```

Difference Between Tuples and Lists in Python

[34]:
```
# a list object occupies more memory than a tuple object
import sys
a_list = ['abc', 'xyz', 123, 231, 13.31, 0.1312]
a_tuple = ('abc', 'xyz', 123, 231, 13.31, 0.1312)
print('The list size:', sys.getsizeof(a_list), 'bytes')
print('The tuple size:', sys.getsizeof(a_tuple), 'bytes')
```

```
The list size: 104 bytes
The tuple size: 88 bytes
```

[36]:
```
#In addition to occupying less memory, processing tuple objects is much faster␣
 ↪than lists,
import timeit
print(timeit.timeit('x=(1,2,3,4,5,6,7,8,9,10,11,12)'))
print(timeit.timeit('x=[1,2,3,4,5,6,7,8,9,10,11,12]'))
```

```
0.024328699999387027
0.09889449999991484
```

[38]:
```
#Some tuples can be used as dictionary keys specifically, tuples that contain␣
 ↪immutable values
# Lists can never be used as dictionary keys, because lists are not immutable
```

```
bigramsTupleDict = {('this','is'):23}
print(bigramsTupleDict )

bigramsTupleDict = {['this','is']:23}
print(bigramsTupleDict )
```

```
{('this', 'is'): 23}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-38-9395ff866797> in <module>
      5 print(bigramsTupleDict )
      6
----> 7 bigramsTupleDict = {['this','is']:23}
      8 print(bigramsTupleDict )

TypeError: unhashable type: 'list'
```

[40]:
```
#Tuples can be used as values in sets whereas lists can not

tupleSet={('this','is'),('Nitheesh',)}
print(tupleSet)

ListSet={['this','is'],['Nitheesh']}
print(ListSet)
```

```
{('this', 'is'), ('Nitheesh',)}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-40-0e9d4c7b5661> in <module>
      4 print(tupleSet)
      5
----> 6 ListSet={['this','is'],['Nitheesh']}
      7 print(ListSet)

TypeError: unhashable type: 'list'
```

[43]:
```
#Due to the immutability, copying tuples and lists are different.
#We need to more careful when copying lists since they are mutable.
#However, we should not have the same concern with tuples because they are␣
 ↪immutable.
#When you copy a tuple and assign it to a new variable, they all point to the␣
 ↪same values in the memory.

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

```
b = a
c = a[:]

print(c)
```

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

If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

Python Interview Questions on Tuples

What are tuples in Python ?

What are characterstics/features of tuples in Python ?

How do you initialize a tuple in Python ?

What is the difference between a Python Tuple and Python list ?

Which one is faster list or tuple ?

Why use a tuple instead of a list ?

To-Do:

Write a Python program to unpack a tuple in several variables

Write a Python program to add an item in a tuple.

Write a Python program to convert a tuple to a string

Sort a tuple of tuples by 2nd item

Write a program to accept five numbers from the user and store it in a tuple