How to Concatenate tuples to nested tuples

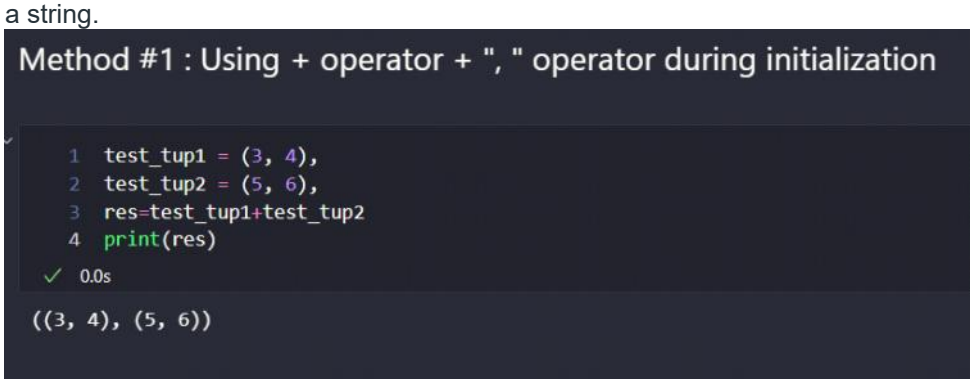
23 June 2025 09:03 AM

Sometimes, while working with tuples, we can have a problem in which we need to convert individual records into a nested collection yet remaining as separate element. Usual addition of tuples, generally adds the contents and hence flattens the resultant container, this is usually undesired. Let's discuss certain ways in which this problem is solved.

Method #1 : Using + operator + ", " operator during initialization

In this method, we perform the usual addition of tuple elements, but while initializing tuples, we add a comma after the tuple so that they don't get flattened while addition.

1. Two tuples are initialized using the comma notation and assigned to variables test_tup1 and test_tup2 respectively.
2. The original tuples are printed using the print function and string concatenation with the help of the str function to convert the tuple to a string.
3. The tuples are concatenated using the + operator and assigned to a new variable called res.
4. The result is printed using the print function and string concatenation with the help of the str function to convert the tuple to a string.




```
# Python3 code to demonstrate working of
# Concatenating tuples to nested tuples
# using + operator + ", " operator during initialization

# initialize tuples
test_tup1 = (3, 4),
test_tup2 = (5, 6),

# printing original tuples
print("The original tuple 1 : " + str(test_tup1))
print("The original tuple 2 : " + str(test_tup2))

# Concatenating tuples to nested tuples
# using + operator + ", " operator during initialization
res = test_tup1 + test_tup2

# printing result
print("Tuples after Concatenating : " + str(res))
```

Output :

```
The original tuple 1 : ((3, 4), )
The original tuple 2 : ((5, 6), )
Tuples after Concatenating : ((3, 4), (5, 6))
```

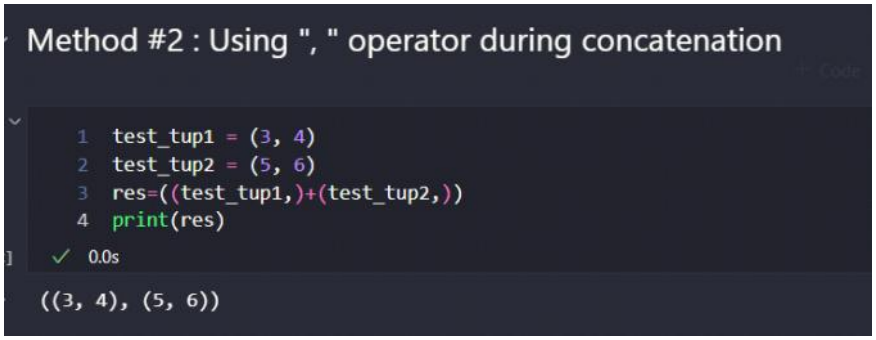
Time complexity: O(1)

Auxiliary space: O(1)

Method #2 : Using ", " operator during concatenation

This task can be performed by applying ", " operator during concatenation as well. It can perform the safe concatenation.

1. First, two tuples test_tup1 and test_tup2 are initialized with values (3, 4) and (5, 6) respectively.
2. The original values of both tuples are printed using the print() function with the help of string concatenation using the + operator.
3. The tuples are concatenated to create a nested tuple using the , operator and assigned to the variable res. Here, the + operator is used to concatenate two tuples and create a new tuple.
4. Finally, the result is printed using the print() function and string concatenation using the + operator. The variable res contains the nested tuple created by concatenating test_tup1 and test_tup2.




```
# Python3 code to demonstrate working of
# Concatenating tuples to nested tuples
# Using ", " operator during concatenation

# initialize tuples
test_tup1 = (3, 4)
test_tup2 = (5, 6)

# printing original tuples
print("The original tuple 1 : " + str(test_tup1))
print("The original tuple 2 : " + str(test_tup2))

# Concatenating tuples to nested tuples
# Using ", " operator during concatenation
res = ((test_tup1, ) + (test_tup2, ))

# printing result
print("Tuples after Concatenating : " + str(res))
```

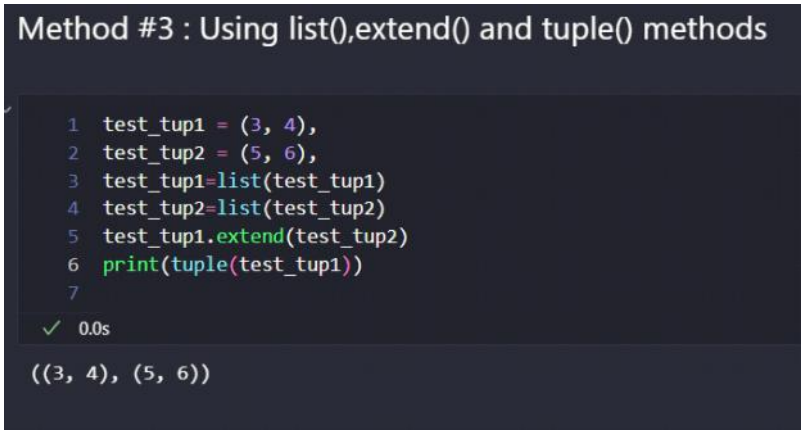
Output :

```
The original tuple 1 : ((3, 4), )
The original tuple 2 : ((5, 6), )
Tuples after Concatenating : ((3, 4), (5, 6))
```

Time complexity: O(1) (constant time)

Auxiliary space: O(1) (constant space)

Method #3 : Using list(),extend() and tuple() methods




```
# Python3 code to demonstrate working of
# Concatenating tuples to nested tuples
# using + operator + ", " operator during initialization

# initialize tuples
test_tup1 = (3, 4),
test_tup2 = (5, 6),

# printing original tuples
print("The original tuple 1 : " + str(test_tup1))
print("The original tuple 2 : " + str(test_tup2))

# Concatenating tuples to nested tuples
test_tup1 = list(test_tup1)
test_tup2 = list(test_tup2)
test_tup1.extend(test_tup2)
# printing result
print("Tuples after Concatenating : " + str(tuple(test_tup1)))
```

Output

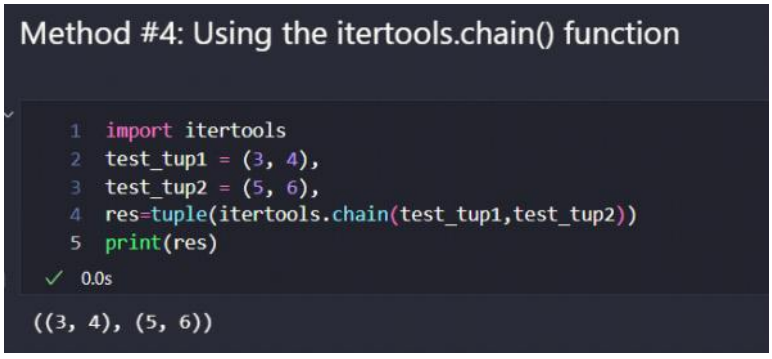
```
The original tuple 1 : ((3, 4),)
The original tuple 2 : ((5, 6),)
Tuples after Concatenating : ((3, 4), (5, 6))
```

Time complexity: O(1), which means it's a constant time operation.

Auxiliary space: O(n), where n is the size of the concatenated tuple.

Method #4: Using the itertools.chain() function

*The **itertools.chain()** function takes multiple iterables and returns a single iterator that yields all the elements from each of the iterables in sequence. By passing the two tuples as arguments to **itertools.chain()**, we can concatenate them into a single tuple, which can then be converted to a nested tuple using the **tuple()** function.*




```
import itertools

test_tup1 = (3, 4),
test_tup2 = (5, 6),

# using itertools.chain() to concatenate tuples to nested tuples
res = tuple(itertools.chain(test_tup1, test_tup2))

# printing result
print("Tuples after Concatenating : ", res)
```

Output

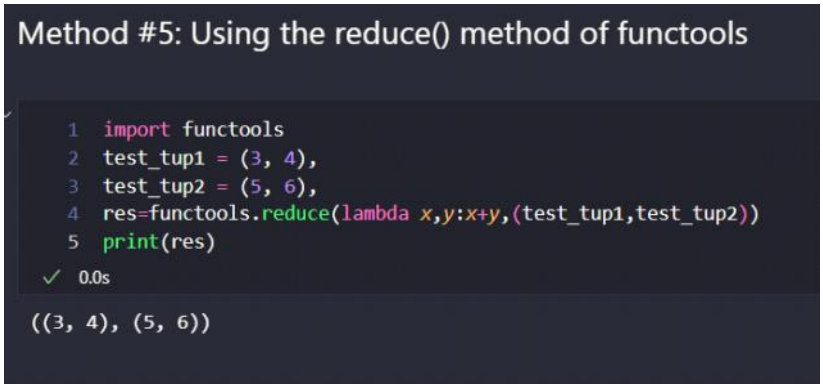
```
Tuples after Concatenating :  ((3, 4), (5, 6))
```

Time complexity: O(n) where n is the total number of elements in both tuples.

Auxiliary space: O(n), where n is the total number of elements in both tuples.

Method #5: Using the reduce() method of functools

This program demonstrates how to concatenate two tuples into a nested tuple using the reduce() method from the functools module. It initializes two tuples, concatenates them using reduce(), and prints the result.




```
# Python3 code to demonstrate working of
# Concatenating tuples to nested tuples
# using functools.reduce() method

# import functools module
import functools

# initialize tuples
test_tup1 = (3, 4),
test_tup2 = (5, 6),

# printing original tuples
print("The original tuple 1 : " + str(test_tup1))
print("The original tuple 2 : " + str(test_tup2))

# Concatenating tuples to nested tuples
```

```
# using functools.reduce() method
res = functools.reduce(lambda x, y: x + y, (test_tup1, test_tup2))

# printing result
print("Tuples after Concatenating : " + str(res))
```

Output

```
The original tuple 1 : ((3, 4),)
The original tuple 2 : ((5, 6),)
Tuples after Concatenating : ((3, 4), (5, 6))
```

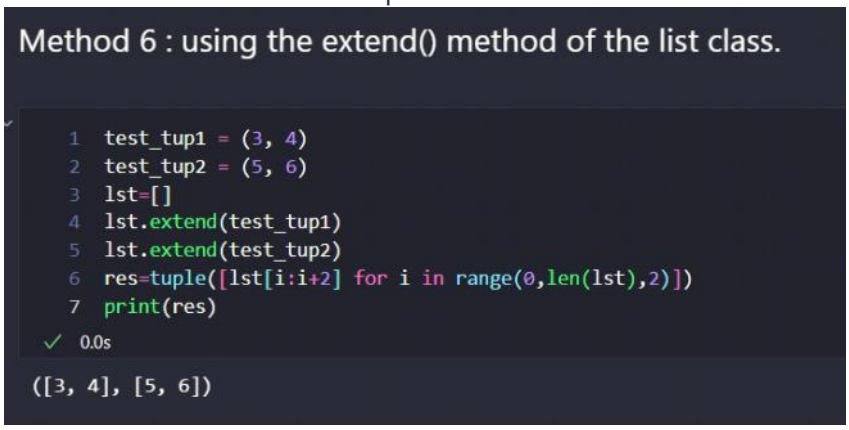
Time complexity: O(n), where n is the total number of elements in the input tuples.

Auxiliary space: O(n), where n is the total number of elements in the input tuples.

Method 6 : using the extend() method of the list class.

Approach:

1. Initialize tuples test_tup1 and test_tup2 with values (3, 4) and (5, 6) respectively.
2. Create an empty list, last.
3. Use the extend() method to add tuples test_tup1 and test_tup2 to last.
4. Use the tuple() constructor to convert lst into a nested tuple.
5. Assign the nested tuple to the variable res.
6. Print the concatenated nested tuple.




```
# Python3 code to demonstrate working of
# Concatenating tuples to nested tuples
# using extend() method of list class

# initialize tuples
test_tup1 = (3, 4)
test_tup2 = (5, 6)

# create an empty list
lst = []

# Concatenating tuples to nested tuples
# using extend() method of list class
lst.extend(test_tup1)
lst.extend(test_tup2)
res = tuple([lst[i:i+2] for i in range(0, len(lst), 2)])

# printing result
print("Tuples after Concatenating : ", res)
```

Output

```
Tuples after Concatenating :  ([3, 4], [5, 6])
```

Time complexity of this method is O(n) where n is the number of elements in both tuples.

Auxiliary space required is O(n) as we are creating a list to store the concatenated tuples.

From <https://www.geeksforgeeks.org/python/python-how-to-concatenate-tuples-to-nested-tuples/>