Python: Tuples

8th Lecture

1. Introduction

Lists aren't the only kind of ordered sequence in Python. The last collection ordered type in Python tuple. Tuples construct simple groups of elements. They work exactly like lists, except that tuples can't be changed in place (they're immutable). This seems to be the first question that always comes up when teaching beginners about tuples: why do we need tuples if we have lists? The best answer, however, seems to be that the immutability of tuples provides some integrity—you can be sure a tuple won't be changed through another reference elsewhere in a program, but there's no such guarantee for lists. Tuples and other immutable, therefore, serve a similar role to "constant" declarations in other languages, though the notion of constantans is associated with objects in Python, not variables. Tuples can also be used in places that lists cannot—for example, as dictionary keys. Some built-in operations may also require or imply tuples instead of lists (e.g., the substitution values in a string format expression).

2. Tuples

A tuple is a type of sequence that very similar to a list, except that, unlike a list, a tuple is immutable; once a tuple is created, you cannot add, delete, replace, and reorder elements. You indicate a tuple literal in Python by written as a series of items in parentheses, not square brackets. Although they don't support as many methods, tuples share most of their properties with lists. Here's a quick look at the basics. Tuples are:

- ***Collections of arbitrary elements***

Like lists, tuples are positional ordered collections of elements (i.e., they maintain a left-to-right order among their contents); like lists, they can embed any kind of elements.

- ***Accessed by offset***

Like lists, elements in a tuple are accessed by offset (not by key); they support all the offset-based access operations, such as indexing and slicing.

- ***Of the category "immutable sequence"***

Like lists, tuples are sequences; they support many of the same operations. However, like strings, tuples are immutable; they don't support any of the in-place change operations applied to lists.

- ***Fixed-length, heterogeneous, and arbitrarily nestable:***

Because tuples are immutable, you cannot change the size of a tuple without making a copy. On the other hand, tuples can hold any type of elements, including other compound objects (e.g., lists, dictionaries, other tuples), and so support arbitrary nesting.

The syntax of tuple as follows:

<tupleName> = (<element$_1$, element$_2$, ….., element$_n$>)

Or

<tupleName> = <element$_1$, element$_2$, ….., element$_n$>

Let's take an example:

```
integerTuple = (11, 22, 33)              # tuple that contents the same data type
intergerTuple= 11, 22, 33                # tuple that contents the same data type
collTuple = ('Address', 'R', 77, 'S', 78)  # tuple that contents the difference data type
strTuple = (["tom", "jerry", "spyke"])   # create a list with strings
strTuple1 = ("tom", "jerry", "spyke")    # create a tuple with strings
integerTuple = ("May", )                 # create a tuple with a single element
```

Note: To create a tuple with a single element, you have to include the ***final comma***:

```
empityTuple=( )                          # this type of tuple called empty tuple
```

Note: If we didn't put any element inside ( ), then the list is called ***empty tuple***.

You can also use other tuples as elements in a tuple, thereby creating a tuple of tuples (***A tuple within another tuple is nested***). Here is one example of such a tuple:

```
nestedTuple = (('R', 'S'), (7, 8), (96,19))       # this tuple of tuple called nested tuple
```

Another way to create a tuple is the built-in function tuple:

tupleName = tuple(<'element, '**>)**

Let's take an example:

```
empityTuple = tuple()                          # this will also create an empty tuple
listTuple = tuple ('1', )                      # create an tuple
istTuple = tuple ([1,2,3,4,4])                 # list from tuple
charTuple = tuple('"spam"')        # create a tuple with characters ('"', 's', 'p', 'a', 'm', '"')
strTuple = tuple(["tom", "jerry", "spyke"])    # create a list with strings
strTuple1 = tuple("tom", "jerry", "spyke")     # Wrong
```

The simple way to **print** a tuple is used the print statement as show in this example:

```
print(tupleName)
```

In this case, it will be printing all elements of tuple including the brackets.

Example 1: Doing these codes in your PC.

```
integerTuple = (11, 22, 33)                    # tuple that contents the same data type
print ("The integer tuple is", integerTuple)
print()
collTuple = ('Address', 'R', 77, 'S', 78)      # tuple that contents the difference data type
print ("The collection tuple is", collTuple)
print()
strTuple = (["tom", "jerry", "spyke"])         # create a list with strings
print ("The string tuple as a list is", strTuple)
print()
strTuple1 = ("tom", "jerry", "spyke")          # create a tuple with strings
print ("The string tuple is", strTuple1)
print()
empityTuple=( )                                # this type of tuple called empty tuple
print ("The empity tuple is", empityTuple)
print()
nestedTuple = (('R', 'S'),(77, 78), (96,19))   # this tuple of tuples called nested tuple
print ("The nested tuple is", nestedTuple)
print()
empityTuple = tuple()                          # this will also create an empty tuple
print ("The empity tuple is", empityTuple)
print()
listTuple = tuple ('1')                        # create an tuple
print ("The content of tuple is", listTuple)
print()
istTuple = tuple ([1,2,3,4,4])                 # list from tuple
print ("The list from tuple is", istTuple)
print()
charTuple = tuple("'spam'")                    # create a tuple with characters ("'", 's', 'p', 'a', 'm', "'")
print ("The tuple with characters is", charTuple)
print()
strTuple = tuple(["tom", "jerry", "spyke"])    # create a list with strings
print ("The string tuple is", strTuple)
print()
strTuple1 = tuple("tom", "jerry", "spyke")     # Wrong
print ("The string tuple is", strTuple1)
```

The output

```
The integer tuple is (11, 22, 33)
The collection tuple is ('Address', 'R', 77, 'S', 78)
The string tuple as a list is ['tom', 'jerry', 'spyke']
The string tuple is ('tom', 'jerry', 'spyke')
The empity tuple is ()
The nested tuple is (('R', 'S'), (77, 78), (96, 19))
The empity tuple is ()
The content of tuple is ('1',)
The list from tuple is (1, 2, 3, 4, 4)
The tuple with characters is ("'", 's', 'p', 'a', 'm', "'")
The string tuple is ('tom', 'jerry', 'spyke')
Traceback (most recent call last):
  File "C:/Users/Raaid
Alubady/AppData/Local/Programs/Python/Python37-
32/tuple.py", line 34, in <module>
 strTuple1 = tuple("tom", "jerry", "spyke")       # Wrong
```

3. Tuples are Immutable

As we mention above, the important difference is that the tuples are **immutable**

Example 2:

```
x = 10, 20, 30, 40, 50        # crate tuple
print (x)
print (x[1:3])
print (x[0])
x[0]=5         # tuple is immutable; cannot update element
print(x)
```

The output

```
10, 20, 30, 40, 50)
(20, 30)
10
Traceback (most recent call last):
File "C:/Users/Raaid  Alubady/AppData/Local/Programs/Python/Python37-
32/tuple.py", line 5, in <module>
 x[0]=5
TypeError: object doesn't support item assignment
```

You can't modify the elements of a tuple, but you can replace one tuple with another:

Example 3:

```
x = 10, 20, 30, 40, 50
print (x)
print (x[1:3])
print (x[0])
x = (5,) + x[1:]
print (x)
```

The output

```
(10, 20, 30, 40, 50)
(20, 30)
10
(5, 20, 30, 40, 50)
```

4. Tuple Assignment

In tuples, there is a special task which is the ability to assign to multiple variables; you can assign to multiple variables at the same time:

$$parameter_1, parameter_2, \dots, parameter_n = value_1, value_2, \dots, value_n$$

Let's take an example:

(x, y, z, m, n) = (10, 20, 30, 40, 50)

(x, y, z, m, n) = ('R', 'S', 'E', 'D', 'M')

(x, y, z, m, n) = ('R', 'S', 'Ellen', (1,2,3), 'May')

Example 4:

```
(x, y, z, m, n) = (10, 20, 30, 40, 50)
print (x, y, z)                                  # printed: 10 20 30
(x, y, z, m, n) = ('R', 'S', 'E', 'D', 'M')
print (x, y, z, m)                               # printed: R S E D
(x, y, z, m, n) = ('R', 'S', 'Ellen', (1,2,3), 'May')
print (x, z, m)                                  # printed: R Ellen (1,2,3)
```

The output
10 20 30
R S E D
R Ellen (1,2,3)

5. Tuple as Return Values

Strictly speaking, a function can only return one value, but if the value is a tuple, the effect is the same as returning multiple values. Here is an example of a function that returns a tuple. The built-in function ***min_max*** that find the largest and smallest elements of a sequence. ***min_max*** computes both and returns a tuple of two values.

Example 5:

```
x = (10, 2, 33, 40, 5)
def min_max(x):
   return min(x), max(x)
print (min_max(x))
```

The output
(2, 40)

6. Traversing a Tuple

The most common way to traverse the elements of a tuple is with **for** loop.

Example 6:

```
t = (6, 9, 8, 7, 0)
print(t)
for elm in range (len(t)):
   print(t[elm], end=" ")
for elm in t:
   print(elm, end=" ")for key in info:
```

The output
(6, 9, 8, 7, 0)
6 9 8 7 0
6 9 8 7 0

7. Tuple Slicing

Slice operator ([start : end]) allows to fetch subtuple from the tuple.

Example 7:

```
t = (6, 9, 8, 7, 0)
print(t[0:5])
print(t[0:3])
print(t[1:3])
print(t[3:5])
print(t[0:0])
```

The output
(6, 9, 8, 7, 0)
(6, 9, 8)
(9, 8)
(7, 0)
()

8. Exercises

1. What are the similarity and difference between List, Dictionary and Tuple?
2. According to your understanding from lecture 8, determine what the built-in functions that are used with a tuple. Build a table that includes these functions with examples?

<Best Regards>

Dr. Raaid Alubady