Python Tuples

Activities

So far you have learned two data-structures: lists, array, series, dataframes and dictionaries. In this class, we will learn Python tuples in detail.

Essentially, Python tuples behave exactly like Python lists with just one difference. Python tuples are immutable in nature. In other words, the contents of a tuple are unchangeable. Once a tuple is created, you cannot add a new item, cannot update an existing item and cannot delete an item.

To sum it up, a Python tuple is an **ordered and immutable** collection of items. A Python list, on the other hand, is an ordered but mutable collection of items.

In real-life situations, a tuple is used to store unchangeable information such as bank account number, email-id, coordinates of a location or any universally true fact such as water has two molecules of hydrogen and one molecule of oxygen.

Activity 1: Create Python Tuple

Let's create a tuple which stores the names of the six naturally occurring noble gases. They are as follows:

`helium (He), neon (Ne), argon (Ar), krypton (Kr), xenon (Xe), radon (Rn)`

To create a tuple, you simply have to write the items enclosed between the common brackets `()` such that the items are separated by a comma.

```
# Student Action: Create a tuple containing the names of the six naturally occurring noble gases.
t_ng=('He','Ne','Ar','Kr','Xe','Rn')
t_ng
```

```
('He', 'Ne', 'Ar', 'Kr', 'Xe', 'Rn')
```

To verify whether a data structure is a tuple or not, you can use the `type()` function.

```
# Student Action: Verify whether the data-structure stored in the 'noble_gases' variable is a tupl
type(t_ng)
```

```
tuple
```

Similar to a Python list, a Python tuple can also contain different types of items such as integer, float, string, list, boolean, tuple etc. Let's learn this concept with the help of an example.

Consider the following properties of helium.

Properties of Helium (He)	Values
Atomic number	2
Atomic mass $\left(\frac{g}{mol}\right)$	4.0026
Electronegativity according to Pauling	unknown
Present in Earth's atmosphere	Yes
Stable isotopes	helium-3, helium-4

The `Values` column in the above list contains items of all the types. Let's put the same items in a tuple and store it in the `helium_props` variable.

```
# Student Action: Create a tuple containing all the properties of Helium.
tr=(2,4.0026,"unknown",True,["helium-3","helium-4"])
print(tr)
```

```
(2, 4.0026, 'unknown', True, ['helium-3', 'helium-4'])
```

As you can see, a tuple can contain different types of items.

Activity 2: Tuple Length

The length of a tuple (or tuple length) is the number of items contained in the tuple. To calculate the length of a tuple (or the number of items present in a tuple), you can use the `len()` function (the same function that you have been using for Python lists).

```
# Student Action: Calculate the number of items contained in the 'helium_props' tuple.
len(tr)
```

```
5
```

The last item `['helium-3', 'helium-4']` is a list acting as a singular item contained in the `helium_props` tuple.

Activity 3: Empty Tuple & One-Item Tuple

You can also create an empty tuple (a tuple having no item) by simply writing the common brackets (or parentheses).

```
# Student Action: Create an empty tuple.
var=()
var
```

```
()
```

Interestingly, creating a tuple containing only one item is quite tricky. As an experiment, create the following five tuples containing only one item and check their types using the `type` function.

```
atomic_num = (2)
atomic_mass = (4.0026)
elec_neg = ('unknown')
in_earth_atmos = (True)
stable_isotopes = (['helium-3', 'helium-4'])
```

```
# Student Action: Create the above five tuples and check their types.
atomic_num = (2)
atomic_mass = (4.0026)
elec_neg = ('unknown')
in_earth_atmos = (True)
stable_isotopes = (['helium-3', 'helium-4'])

type(atomic_num)
```

```
int
```

```
type(atomic_mass)
```

```
float
```

```
type(elec_neg)
```

```
str
```

As you can see, the `type()` function returns all the other types except for `tuple`. This is how the Python interpreter works. We can't do anything about it.

Now, put a comma after the item in each of the above five tuples and check their types again.

```
# Student Action: Create the above five tuples again by putting a comma after item in each tuple a
atomic_num = (2,)
atomic_mass = (4.0026,)
elec_neg = ('unknown',)
in_earth_atmos = (True,)
stable_isotopes = (['helium-3', 'helium-4'],)
```

```
type(atomic_num)
```

```
tuple
```

```
type(atomic_mass)
```

```
tuple
```

```
Start coding or generate with AI.
```

As you can see, we have now created five tuples. Each of them contains only one item. Hence, the trick to create a tuple having only one item is to put a comma after the item.

Note: Even if you don't enclose the items within parentheses but put a trailing comma, then also Python will create a tuple.

```
# Student Action: Create two tuples without putting parentheses: one having only one item and anot
# Verify whether they both are tuples or not.
tr = 1,2,3,4
type(tr)
```

```
tuple
```

```
t=0,
```

```
type(t)
```

```
tuple
```

As you can see, even without putting parentheses, we can create a tuple. This is a very unique property of a Python tuple.

Activity 4: Tuple Indexing^

Tuple indexing is exactly the same as list indexing. Every item in a Python tuple, occupies a unique position called index.

- The first item in a tuple occupies `index = 0`.
- Similarly, the second item occupies `index = 1` and so on.
- The last item occupies `index = (n - 1)`, where `n` is the number of items contained in a tuple.

To get an item of a specific index, write the name of the variable storing the tuple followed by the index value enclosed between square brackets `[]`.

Syntax: `tuple_name[index_value]`

Here's the expected output for the following exercise:

```
Item at index 0 is 2
Item at index 1 is 4.0026
Item at index 2 is unknown
Item at index 3 is True
Item at index 4 is ['helium-3', 'helium-4']
```

```
# Student Action: Print all the items one-by-one contained in the 'helium_props' tuple using the i
for i in range(len(tr)):
  print(i," : ",str(tr[i]))
```

```
0  :  2
1  :  4.0026
2  :  unknown
3  :  True
4  :  ['helium-3', 'helium-4']
```

As you can see, you can get individual items by writing their indices enclosed between square brackets after the variable containing the tuple.

Just like a Python list, you can use negative indexing as well to retrieve an item from a tuple.

Here's the expected output for the following exercise:

```
Item at index -5 is 2
Item at index -4 is 4.0026
Item at index -3 is unknown
Item at index -2 is True
Item at index -1 is ['helium-3', 'helium-4']
```

```
# Student Action: Print all the items one-by-one contained in the 'helium_props' tuple using the n
for i in range(len(tr)):
  print("Item at index", -(len(tr)-i) , " is " , tr[i])
```

```
Item at index -5  is  2
Item at index -4  is  4.0026
Item at index -3  is  unknown
Item at index -2  is  True
Item at index -1  is  ['helium-3', 'helium-4']
```

Here's the expected output for the following exercise:

```
Item at index -1 is ['helium-3', 'helium-4']
Item at index -2 is True
Item at index -3 is unknown
Item at index -4 is 4.0026
Item at index -5 is 2
```

```
# Student Action: Get all the items of the 'helium_props' tuple in the reverse order using the neg
for i in range(len(tr)):
  print("Item at index ",-(i+1),tr[len(tr)-i-1])
```

```
Item at index  -1 ['helium-3', 'helium-4']
Item at index  -2 True
Item at index  -3 unknown
Item at index  -4 4.0026
Item at index  -5 2
```

You can also retrieve all the items from a tuple without using the `range()` function in a `for` loop.

Here's the expected output for the following exercise:

```
Item at index 0 is 2
Item at index 1 is 4.0026
Item at index 2 is unknown
Item at index 3 is True
Item at index 4 is ['helium-3', 'helium-4']
```

```
# Student Action: Retrieve all the items from 'helium_props' tuple without using the 'range()' fun
j=0;
for i in tr:
  print(j , " " ,i);
  j+=1;
```

```
0   2
1   4.0026
2   unknown
3   True
4   ['helium-3', 'helium-4']
```

Start coding or generate with AI.

Start coding or generate with AI.

Note: The `i += 1` is another way of writing `i = i + 1`. Similarly,

- `i -= 1` is another way of writing `i = i - 1`.
- `i *= 2` is another way of writing `i = i * 2` and so on.

Activity 5: Tuple Slicing^^

Again, tuple slicing is exactly the same as the list slicing. We will quickly go through this concept.

```
# Student Action: Get the first four items from the 'helium_props' tuple.
tr[0:4]
```

```
(2, 4.0026, 'unknown', True)
```

Start coding or generate with AI.

Start coding or generate with AI.

```
# Student Action: Get all the items from the 'helium_props' tuple in the reverse order.
tr[::-1]
```

```
(['helium-3', 'helium-4'], True, 'unknown', 4.0026, 2)
```

Start coding or generate with AI.

Start coding or generate with AI.

```
# Student Action: Get the alternate items from the 'helium_props' tuple.
tr[::-2]
```

```
(['helium-3', 'helium-4'], 'unknown', 2)
```

```
print(tr[::2])
```

```
(2, 'unknown', ['helium-3', 'helium-4'])
```

```
tr
```

```
(2, 4.0026, 'unknown', True, ['helium-3', 'helium-4'])
```

```
# Student Action: Get all the items from the 'helium_props' tuple using negative indexing except f
print(tr[1:-1:])
```

```
(4.0026, 'unknown', True)
```

Start coding or generate with AI.

```
# Student Action: Get the second item of the list stored in the 'helium_props' tuple.
tr[len(tr)-1][1]
```

```
'helium-4'
```

Start coding or generate with AI.

Activity 6: The `index()` Function

You can use the `index()` function to find out the index of an item in a tuple. The same function can also be used for a Python list.

Syntax: `tuple_name.index(item)`

```
# Student Action: Get the index of the item 'unknown' that is present in the 'helium_props' tuple.
unk_ind = tr.index("unknown")
unk_ind
```

```
2
```

Note: The `index()` function throws `ValueError` if an item does not exist in the tuple.

Activity 7: The `count()` Function

The `count()` function in tuple counts the number of times an item occurs in a tuple. The same function can also be used for a Python list.

Here we have a tuple containing the FIFA football world cup winners starting from the year 1982 till 2018.

`fifa_wc_winners = ('Italy', 'Argentina', 'Germany', 'Brazil', 'France', 'Brazil', 'Italy', 'Spain', 'Germany', 'France')`

Find out how many times France and Brazil have won the world cup in the given period.

```
# Student Action: Count the number of times 'France' & 'Brazil' have won the world cup between the
fifa_wc_winners = ('Italy', 'Argentina', 'Germany', 'Brazil', 'France', 'Brazil', 'Italy', 'Spain'
fifa_wc_winners.count("France")
fifa_wc_winners.count("Brazil")
```

```
2
```

So, both the countries have won the FIFA world cup twice between the years 1982 and 2018. This also proves that a tuple can contain repeated elements (or items).

Activity 8: Add, Replace & Delete Item^^^

As discussed at the beginning of this class, we cannot add, replace (or update) & delete an item from a tuple. Let's verify this theory.

Let's try to replace the item `'unknown'` contained in the `helium_props` tuple with the string `'Not Available'`. We should get an error.

```
# Student Action: Try to replace the item 'unknown' contained in the 'helium_props' tuple with the
tr[2] = "Not Available"
tr[2]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-62-08aecd4ad1a3> in <module>
      1 # Student Action: Try to replace the item 'unknown' contained in the 'helium_props' tuple with the string 'Not Available'.
----> 2 tr[2] = "Not Available"
      3 tr[2]

TypeError: 'tuple' object does not support item assignment
```

As you can see, Python throws `TypeError` saying that `'tuple' object does not support item assignment`. Hence, we have verified that we cannot replace or update an existing item in a tuple.

Let's try to delete the item `'unknown'` using the `del` keyword. It works for a Python list as well. There exists no `remove()` function for a Python tuple like the one that exists for a Python list.

Note: Again you will get an error.

```
# Student Action: Try to delete the item 'unknown' from the 'helium_props' tuple. using the 'del'
del tr

tr
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-9b8eec1fd974> in <cell line: 1>()
----> 1 tr

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

As you can see, Python throws `TypeError` saying that `'tuple' object doesn't support item deletion`. Hence, we have verified that we cannot delete an existing item from a tuple.

There exists no function like the `append()` function (exists for a Python list) that can add an item to a tuple. But can concatenate or join two or more tuples.