What is Tuple in Python?

Tuple data type in Python is a collection of various immutable Python objects separated by commas. Tuples are much similar to Python Lists, but they are syntactically different, i.e., in lists we use square brackets while in tuples we use parentheses. In this module, we will learn all about the tuple data type in order to get started with it.

Tuples are ordered collections of heterogeneous data that are unchangeable. Heterogeneous means tuple can store variables of all types.

Tuple has the following characteristics

- **Ordered**: Tuples are part of sequence data types, which means they hold the order of the data insertion. It maintains the index value for each item.
- **Unchangeable**: Tuples are unchangeable, which means that we cannot add or delete items to the tuple after creation.
- **Heterogeneous**: Tuples are a sequence of data of different data types (like integer, float, list, string, etc;) and can be accessed through indexing and slicing.
- **Contains Duplicates**: Tuples can contain duplicates, which means they can have items with the same value.

Creating a Tuple

We can create a tuple using the two ways

1. Using parenthesis (): A tuple is created by enclosing comma-separated items inside rounded brackets.
2. Using a tuple() constructor: Create a tuple by passing the comma-separated items inside the tuple().

A tuple can have items of different data type integer, float, list, string, etc;

In [1]:
```python
# create a tuple using ()
# number tuple
number_tuple = (10, 20, 25.75)
print(number_tuple)
# Output (10, 20, 25.75)

# string tuple
string_tuple = ('Deepali', 'Nikita', 'Pooja')
print(string_tuple)
# Output (''Deepali', 'Nikita', 'Pooja')

# mixed type tuple
sample_tuple = ('Deepali', 30, 45.75, [25, 78])
print(sample_tuple)
# Output ('Deepali', 30, 45.75, [25, 78])

# create a tuple using tuple() constructor
sample_tuple2 = tuple(('Deepali', 30, 45.75, [23, 78]))
print(sample_tuple2)
# Output ('Deepali', 30, 45.75, [23, 78])
```

```
(10, 20, 25.75)
('Deepali', 'Nikita', 'Pooja')
('Deepali', 30, 45.75, [25, 78])
('Deepali', 30, 45.75, [23, 78])
```

Packing and Unpacking

A tuple can also be created without using a tuple() constructor or enclosing the items inside the parentheses. It is called the variable “Packing.”

In Python, we can create a tuple by packing a group of variables. Packing can be used when we want to collect multiple values in a single variable. Generally, this operation is referred to as tuple packing.

Similarly, we can unpack the items by just assigning the tuple items to the same number of variables. This process is called “Unpacking.”

In [7]:
```python
# packing variable into tuple

tup1 = 1,2,"hello"
print(tup1)

print(type(tup1))

# unpacking variable into tuple

a,b,c=tup1
print(tup1)


```

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

As we can see in the above output, three tuple items are assigned to individual variables i, j, k, respectively. In case we assign fewer variables than the number of items in the tuple, we will get the value error with the message too many values to unpack

Length of a Tuple

We can find the length of the tuple using the len() function. This will return the number of items in the tuple.

In [8]:
```python
tup = ["p","y","t","h","o","n"]
print(len(tup))
```

```
6
```

Iterating a Tuple

We can iterate a tuple using a for loop

In [9]:
```python
# create a tuple
tup1 = (1,2,3,"hi",[4,5,6])

# iterate a tuple
for i in tup1:
    print(i)
```

```
1
2
3
hi
[4, 5, 6]
```

Accessing items of a Tuple

Tuple can be accessed through indexing and slicing.

- Using indexing, we can access any item from a tuple using its index number
- Using slicing, we can access a range of items from a tuple

Indexing

A tuple is an ordered sequence of items, which means they hold the order of the data insertion. It maintains the index value for each item.

We can access an item of a tuple by using its index number inside the index operator [] and this process is called “Indexing”.

Note:

- As tuples are ordered sequences of items, the index values start from 0 to the tuple’s length.
- Whenever we try to access an item with an index more than the tuple’s length, it will throw the 'Index Error'.
- Similarly, the index values are always integer. If we give any other type, then it will throw Type Error.

In [10]:
```
tup = ["D","E","E","P","A","L","I"]

for i in range(4):
    print(tup[i])
```

```
D
E
E
P
```

As seen in the above example, we print the tuple’s first four items with the indexing.

Note: If we mention the index value greater than the length of a tuple then it will throw an index error.

In [12]:
```
tup = ["D","E","E","P","A","L","I"]
print(tup[7])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-12-51191cf78f71> in <module>
      1 tup = ["D","E","E","P","A","L","I"]
----> 2 print(tup[7])

IndexError: list index out of range
```

Also, if you mention any index value other than integer then it will throw Type Error.

In [13]:
```python
tuple1 = ('P', 'Y', 'T', 'H', 'O', 'N')

# TypeError: tuple indices must be integers or slices, not float
print(tuple1[2.0])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-3807a85d9512> in <module>
      2 
      3 # TypeError: tuple indices must be integers or slices, not float
----> 4 print(tuple1[2.0])

TypeError: tuple indices must be integers or slices, not float
```

Negative Indexing

The index values can also be negative, with the last but the first items having the index value as -1 and second last -2 and so on.

For example, We can access the last item of a tuple using tuple_name[-1].

Let's do two things here

- Access tuple items using the negative index value
- Iterate tuple using negative indexing

In [14]:
```
tuple1 = ('P', 'Y', 'T', 'H', 'O', 'N')
# Negative indexing
# print last item of a tuple
print(tuple1[-1])  # N
# print second last
print(tuple1[-2])  # O

# iterate a tuple using negative indexing
for i in range(-6, 0):
    print(tuple1[i], end=", ")
# Output P, Y, T, H, O, N,
```

```
N
O
P, Y, T, H, O, N,
```

Slicing a tuple

We can even specify a range of items to be accessed from a tuple using the technique called ‘Slicing.’ The operator used is ':'.

We can specify the start and end values for the range of items to be accessed from the tuple. The output will be a tuple, and it includes the range of items with the index values from the start till the end of the range. The end value item will be excluded.

We should keep in mind that the index value always starts with a 0.

For easy understanding, we will be using an integer tuple with values from 0 to 9 similar to how an index value is assigned.

In [15]:
```
tuple1 = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

# slice a tuple with start and end index number
print(tuple1[1:5])
# Output (1, 2, 3, 4)
```

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

As seen in the above output the values starting from 1 to 4 are printed. Here the last value in the range 5 is excluded.

Note:

- If the start value is not mentioned while slicing a tuple, then the values in the tuples start from the first item until the end item in the range. Again the end item in the range will be excluded.
- Similarly, we can mention a slicing range without the end value. In that case, the item with the index mentioned in the start value of the range till the end of the tuple will be returned.

In [16]:
```python
tuple1 = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

# slice a tuple without start index
print(tuple1[:5])
# Output (0, 1, 2, 3, 4)

# slice a tuple without end index
print(tuple1[6:])
# Output (6, 7, 8, 9, 10)
```

```
(0, 1, 2, 3, 4)
(6, 7, 8, 9, 10)
```

Similarly, we can slice tuple using negative indexing as well. The last but first item will have the index -1.

Here we can see that the items with the negative indexes starting from -1 till -4 are printed excluding -5.

Finding an item in a Tuple

We can search for a certain item in a tuple using the index() method and it will return the position of that particular item in the tuple.

The index() method accepts the following three arguments

1. item – The item which needs to be searched
2. start – (Optional) The starting value of the index from which the search will start
3. end – (Optional) The end value of the index search

In [17]:
```python
tuple1 = (10, 20, 30, 40, 50)

# get index of item 30
position = tuple1.index(30)
print(position)

```

```
2
```

Find within a range

We can mention the start and end values for the index() method so that our search will be limited to those values.

In [18]:
```python
tuple1 = (10, 20, 30, 40, 50, 60, 70, 80)
# Limit the search locations using start and end
# search only from location 4 to 6
# start = 4 and end = 6
# get index of item 60
position = tuple1.index(60, 4, 6)
print(position)

```

```
5
```

Checking if an item exists

We can check whether an item exists in a tuple by using the in operator. This will return a boolean True if the item exists and False if it doesn't.

In [19]:
```python
tuple1 = (10, 20, 30, 40, 50, 60, 70, 80)
# checking whether item 50 exists in tuple
print(50 in tuple1)
# Output True
print(500 in tuple1)
# Output False
```

```
True
False
```

Adding and changing items in a Tuple

A list is a mutable type, which means we can add or modify values in it, but tuples are immutable, so they cannot be changed.

Also, because a tuple is immutable there are no built-in methods to add items to the tuple.

If you try to modify the value you will get an error.

As a workaround solution, we can convert the tuple to a list, add items, and then convert it back to a tuple. As tuples are ordered collection like lists the items always get added in the end.

In [20]:
```python
tup1 = (1,2,3,4,5)

#coverting tuple into list
a = list(tup1)
a.append(6)

# converting list back into tuple
tup1=tuple(a)
print(tup1)
```

```
(1, 2, 3, 4, 5, 6)
```

Modify nested items of a tuple

One thing to remember here, If one of the items is itself a mutable data type as a list, then we can change its values in the case of a nested tuple.

For example, let’s assume you have the following tuple which has a list as its last item and you wanted to modify the list items.

In [21]:
```
tup = (5,6,7,[8,9,10])
print(tup)

# modifing last item first value:
tup[3][0] = 89
print(tup)
```

```
(5, 6, 7, [8, 9, 10])
(5, 6, 7, [89, 9, 10])
```

In [22]:
```
tuple1 = (0, 1, 2, 3, 4, 5)

# converting tuple into a list
sample_list = list(tuple1)
# modify 2nd item
sample_list[1] = 10

# converting list back into a tuple
tuple1 = tuple(sample_list)
print(tuple1)
# Output (0, 10, 2, 3, 4, 5)
```

```
(0, 10, 2, 3, 4, 5)
```

Removing items from a tuple

Tuples are immutable so there are no pop() or remove() methods for the tuple. We can remove the items from a tuple using the following two ways.

1. Using del keyword
2. By converting it into a list

Using del keyword

The del keyword will delete the entire tuple.

In [26]:
```python
a = (2,3,4,5,6)
del a
print(a)  #we are getting error when we try to access a deleted tuple.
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-26-7117a28423d5> in <module>
      1 a = (2,3,4,5,6)
      2 del a
----> 3 print(a)

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

By converting it into a List

In [27]:
```python
tuple1 = (0, 1, 2, 3, 4, 5)

# converting tuple into a list
sample_list = list(tuple1)
# reomve 2nd item
sample_list.remove(2)

# converting list back into a tuple
tuple1 = tuple(sample_list)
print(tuple1)
# Output (0, 1, 3, 4, 5)
```

```
(0, 1, 3, 4, 5)
```

Count the occurrence of an item in a tuple

a tuple can contain duplicate items. To determine how many times a specific item occurred in a tuple, we can use the count() method of a tuple object.

The count() method accepts any value as a parameter and returns the number of times a particular value appears in a tuple.

In [28]:
```python
tuple1 = (10, 20, 60, 30, 60, 40, 60)
# Count all occurrences of item 60
count = tuple1.count(60)
print(count)
# Output 3

count = tuple1.count(600)
print(count)
# Output 0
```

```
3
0
```

Concatenating two Tuples

We can concatenate two or more tuples in different ways. One thing to note here is that tuples allow duplicates, so if two tuples have the same item, it will be repeated twice in the resultant tuple.

Using the + operator

We can add two tuples using the + operator. This is a very and straightforward method and the resultant tuple will have items from both the tuples.

In [29]:
```python
tuple1 = (1, 2, 3, 4, 5)
tuple2 = (3, 4, 5, 6, 7)

# concatenate tuples using + operator
tuple3 = tuple1 + tuple2
print(tuple3)
# Output (1, 2, 3, 4, 5, 3, 4, 5, 6, 7)
```

```
(1, 2, 3, 4, 5, 3, 4, 5, 6, 7)
```

Using the sum() function

We can also use the Python built-in function sum to concatenate two tuples. But the sum function of two iterables like tuples always needs to start with Empty Tuple.

In [30]:
```python
tuple1 = (1, 2, 3, 4, 5)
tuple2 = (3, 4, 5, 6, 7)

# using sum function
tuple3 = sum((tuple1, tuple2), ())
print(tuple3)
# Output (1, 2, 3, 4, 5, 3, 4, 5, 6, 7)
```

```
(1, 2, 3, 4, 5, 3, 4, 5, 6, 7)
```

Nested tuples

Nested tuples are tuples within a tuple i.e., when a tuple contains another tuple as its member then it is called a nested tuple.

In order to retrieve the items of the inner tuple we need a nested for loop

In [31]:
```python
nest_tup = ((1,2,3),(4,5,6),"python")

# access the first item of the third tuple
print(nest_tup[2][0])

# iterate a nested tuple
for i in nest_tup:
    print("tuple",i,"element")
    for j in i:
        print(j,end=",")
    print("\n")
```

```
p
tuple (1, 2, 3) element
1,2,3,

tuple (4, 5, 6) element
4,5,6,

tuple python element
p,y,t,h,o,n,
```

Use built-in functions with tuple

min() and max()

As the name suggests the max() function returns the maximum item in a tuple and min() returns the minimum value in a tuple.

In [32]:

```python
tuple1 = ('xyz', 'zara', 'abc')
# The Maximum value in a string tuple
print(max(tuple1))
# Output zara

# The minimum value in a string tuple
print(min(tuple1))
# Output abc

tuple2 = (11, 22, 10, 4)
# The Maximum value in a integer tuple
print(max(tuple2))
# Output 22
# The minimum value in a integer tuple
print(min(tuple2))
# Output 4
```

```
zara
abc
22
4
```

Note: We can’t find the max() and min() for a heterogeneous tuple (mixed types of items). It will throw Type Error

In [33]:

```python
tuple3 = ('a', 'e', 11, 22, 15)
# max item
print(max(tuple3))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-33-3393ac6a570b> in <module>
      1 tuple3 = ('a', 'e', 11, 22, 15)
      2 # max item
----> 3 print(max(tuple3))

TypeError: '>' not supported between instances of 'int' and 'str'
```