Tuples: Basic tuples Operations, Accessing, Inserting, Deleting, Updating elements, Built-in tuple Functions & Methods.

Python Tuples

A comma-separated group of items is called a Python triple. The ordering, settled items, and reiterations of a tuple are to some degree like those of a rundown, but in contrast to a rundown, a tuple is unchanging.

The main difference between the two is that we cannot alter the components of a tuple once they have been assigned. On the other hand, we can edit the contents of a list.

Example

1. ("Suzuki", "Audi", "BMW"," Skoda ") is a tuple.

Features of Python Tuple

- Tuples are an immutable data type, meaning their elements cannot be changed after they are generated.
- Each element in a tuple has a specific order that will never change because tuples are ordered sequences.

Forming a Tuple:

All the objects-also known as "elements"-must be separated by a comma, enclosed in parenthesis (). Although parentheses are not required, they are recommended.

PauseNext
Unmute

Current TimeÂ 0:06

/

DurationÂ 18:10
Loaded: 4.04%
Â
Fullscreen

Any number of items, including those with various data types (dictionary, string, float, list, etc.), can be contained in a tuple.

Code

```
# Python program to show how to create a tuple
# Creating an empty tuple
```

```python
empty_tuple = ()
print("Empty tuple: ", empty_tuple)

# Creating tuple having integers
int_tuple = (4, 6, 8, 10, 12, 14)
print("Tuple with integers: ", int_tuple)

# Creating a tuple having objects of different data types
mixed_tuple = (4, "Python", 9.3)
print("Tuple with different data types: ", mixed_tuple)

# Creating a nested tuple
nested_tuple = ("Python", {4: 5, 6: 2, 8:2}, (5, 3, 5, 6))
print("A nested tuple: ", nested_tuple)
```

Output:

```
Empty tuple:  ()
Tuple with integers:  (4, 6, 8, 10, 12, 14)
Tuple with different data types:  (4, 'Python', 9.3)
A nested tuple:  ('Python', {4: 5, 6: 2, 8: 2}, (5, 3, 5, 6))
```

Parentheses are not necessary for the construction of multiples. This is known as triple pressing.

Code

```python
# Python program to create a tuple without using parentheses
# Creating a tuple
tuple_ = 4, 5.7, "Tuples", ["Python", "Tuples"]
# Displaying the tuple created
print(tuple_)
# Checking the data type of object tuple_
print(type(tuple_) )
# Trying to modify tuple_
try:
    tuple_[1] = 4.2
except:
    print(TypeError )
```

Output:

```
(4, 5.7, 'Tuples', ['Python', 'Tuples'])
<class 'tuple'>
```

```
<class 'TypeError'>
```

The development of a tuple from a solitary part may be complex.

Essentially adding a bracket around the component is lacking. A comma must separate the element to be recognized as a tuple.

Code

```
# Python program to show how to create a tuple having a single element
single_tuple = ("Tuple")
print( type(single_tuple) )
# Creating a tuple that has only one element
single_tuple = ("Tuple",)
print( type(single_tuple) )
# Creating tuple without parentheses
single_tuple = "Tuple",
print( type(single_tuple) )
```

Output:

```
<class 'str'>
<class 'tuple'>
<class 'tuple'>
```

Accessing Tuple Elements

A tuple's objects can be accessed in a variety of ways.

Indexing

Indexing We can use the index operator [] to access an object in a tuple, where the index starts at 0.

The indices of a tuple with five items will range from 0 to 4. An Index Error will be raised assuming we attempt to get to a list from the Tuple that is outside the scope of the tuple record. An index above four will be out of range in this scenario.

Because the index in Python must be an integer, we cannot provide an index of a floating data type or any other type. If we provide a floating index, the result will be TypeError.

The method by which elements can be accessed through nested tuples can be seen in the example below.

ADVERTISEMENT

Code

```python
# Python program to show how to access tuple elements
# Creating a tuple
tuple_ = ("Python", "Tuple", "Ordered", "Collection")
print(tuple_[0])
print(tuple_[1])
# trying to access element index more than the length of a tuple
try:
    print(tuple_[5])
except Exception as e:
    print(e)
# trying to access elements through the index of floating data type
try:
    print(tuple_[1.0])
except Exception as e:
    print(e)
# Creating a nested tuple
nested_tuple = ("Tuple", [4, 6, 2, 6], (6, 2, 6, 7))

# Accessing the index of a nested tuple
print(nested_tuple[0][3])
print(nested_tuple[1][1])
```

Output:

```
Python
Tuple
tuple index out of range
tuple indices must be integers or slices, not float
l
6
```

- **Negative Indexing**

Python's sequence objects support negative indexing.

The last thing of the assortment is addressed by - 1, the second last thing by - 2, etc.

Code

```python
# Python program to show how negative indexing works in Python tuples
# Creating a tuple
tuple_ = ("Python", "Tuple", "Ordered", "Collection")
```

4. # Printing elements using negative indices
5. print("Element at -1 index: ", tuple_[-1])
6. print("Elements between -4 and -1 are: ", tuple_[-4:-1])

Output:

```
Element at -1 index:  Collection
Elements between -4 and -1 are:  ('Python', 'Tuple', 'Ordered')
```

Append elements in Tuple

Tuple is immutable, although you can use the + operator to concatenate several tuples. The old object is still present at this point, and a new object is created.

Example

Following is an example to append the tuple −

```
s=(2,5,8)
s_append = s + (8, 16, 67)
print(s_append)
print(s)
```

Output

Following is an output of the above code −

```
(2, 5, 8, 8, 16, 67)
(2, 5, 8)
```

Note− Concatenation is only possible with tuples. It can't be concatenated to other kinds, such lists.

Example

Following is an example of concatenating a tuple with a list −

```
s=(2,5,8)
s_append = (s + [8, 16, 67])
print(s_append)
print(s)
```

Output

The following error came as an output of the above code −

```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    s_append = (s + [8, 16, 67])
TypeError: can only concatenate tuple (not "list") to tuple
```

Concatenating Tuple with one element

You can concatenate a tuple with one element if you want to add an item to it.

Example

Following is an example to concatenate a tuple with one element −

```
s=(2,5,8)
s_append_one = s + (4,)
print(s_append_one)
```

Output

Following is an output of the above code.

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

Note − The end of a tuple with only one element must contain a comma as seen in the above example.

Adding/Inserting items in a Tuple

You can concatenate a tuple by adding new items to the beginning or end as previously mentioned; but, if you wish to insert a new item at any location, you must convert the tuple to a list.

Example

Following is an example of adding items in tuple −

```
s= (2,5,8)
# Python conversion for list and tuple to one another
x = list(s)
print(x)
print(type(x))

# Add items by using insert ()
x.insert(5, 90)
print(x)
```

```
# Use tuple to convert a list to a tuple ().
s_insert = tuple(x)
print(s_insert)
print(type(s_insert))
```

Output

we get the following output of the above code.

```
[2, 5, 8]
⌧class 'list'>
[2, 5, 8, 90]
(2, 5, 8, 90)
⌧class 'tuple'>
```

Using append() method

A new element is added to the end of the list using the **append() method**.

Example

Following is an example to append an element using append() method −

```
# converting tuple to list
t=(45,67,36,85,32)
l = list(t)
print(l)
print(type(l))

# appending the element in a list
l.append(787)
print(l)

# Converting the list to tuple using tuple()
t=tuple(l)
print(t)
```

Output

Following is an output of the above code

```
[45, 67, 36, 85, 32]
⌧class 'list'>
[45, 67, 36, 85, 32, 787]
(45, 67, 36, 85, 32, 787)
```

Slicing

Tuple slicing is a common practice in Python and the most common way for programmers to deal with practical issues. Look at a tuple in Python. Slice a tuple to access a variety of its elements. Using the colon as a straightforward slicing operator (:) is one strategy.

To gain access to various tuple elements, we can use the slicing operator colon (:).

Code

```
# Python program to show how slicing works in Python tuples
# Creating a tuple
tuple_ = ("Python", "Tuple", "Ordered", "Immutable", "Collection", "Objects")
# Using slicing to access elements of the tuple
print("Elements between indices 1 and 3: ", tuple_[1:3])
# Using negative indexing in slicing
print("Elements between indices 0 and -4: ", tuple_[:-4])
# Printing the entire tuple by using the default start and end values.
print("Entire tuple: ", tuple_[:])
```

Output:

```
Elements between indices 1 and 3:  ('Tuple', 'Ordered')
Elements between indices 0 and -4:  ('Python', 'Tuple')
Entire tuple:  ('Python', 'Tuple', 'Ordered', 'Immutable', 'Collection', 'Objects')
```

Deleting a Tuple

A tuple's parts can't be modified, as was recently said. We are unable to eliminate or remove tuple components as a result.

However, the keyword del can completely delete a tuple.

Code

```
# Python program to show how to delete elements of a Python tuple
# Creating a tuple
tuple_ = ("Python", "Tuple", "Ordered", "Immutable", "Collection", "Objects")
# Deleting a particular element of the tuple
try:
    del tuple_[3]
    print(tuple_)
except Exception as e:
    print(e)
```

```
# Deleting the variable from the global space of the program
del tuple_
# Trying accessing the tuple after deleting it
try:
    print(tuple_)
except Exception as e:
    print(e)
```

Output:

```
'tuple' object does not support item deletion
name 'tuple_' is not defined
```

Update element :

# Original tuple

original_tuple = (1, 2, 3, 4, 5)

# Index of the element to be updated

index_to_update = 2

# New value to replace the old value

new_value = 10

# Create a new tuple with the updated element

updated_tuple = original_tuple[:index_to_update] + (new_value,) + original_tuple[index_to_update + 1:]

print(updated_tuple)

Repetition Tuples in Python

Code

```
# Python program to show repetition in tuples
tuple_ = ('Python',"Tuples")
print("Original tuple is: ", tuple_)
# Repeting the tuple elements
```

```
tuple_ = tuple_ * 3
print("New tuple is: ", tuple_)
```

Output:

```
Original tuple is:  ('Python', 'Tuples')
New tuple is:  ('Python', 'Tuples', 'Python', 'Tuples', 'Python', 'Tuples')
```

Tuple Methods

Like the list, Python Tuples is a collection of immutable objects. There are a few ways to work with tuples in Python. With some examples, this essay will go over these two approaches in detail.

The following are some examples of these methods.

- **Count () Method**

The times the predetermined component happens in the Tuple is returned by the count () capability of the Tuple.

Code

```
# Creating tuples
T1 = (0, 1, 5, 6, 7, 2, 2, 4, 2, 3, 2, 3, 1, 3, 2)
T2 = ('python', 'java', 'python', 'Tpoint', 'python', 'java')
# counting the appearance of 3
res = T1.count(2)
print('Count of 2 in T1 is:', res)
# counting the appearance of java
res = T2.count('java')
print('Count of Java in T2 is:', res)
```

Output:

```
Count of 2 in T1 is: 5
Count of java in T2 is: 2
```

Index() Method:

The Index() function returns the first instance of the requested element from the Tuple.

Parameters:

- The thing that must be looked for.

- Start: (Optional) the index that is used to begin the final (optional) search: The most recent index from which the search is carried out
- Index Method

Code

```
# Creating tuples
Tuple_data = (0, 1, 2, 3, 2, 3, 1, 3, 2)
# getting the index of 3
res = Tuple_data.index(3)
print('First occurrence of 1 is', res)
# getting the index of 3 after 4th
# index
res = Tuple_data.index(3, 4)
print('First occurrence of 1 after 4th index is:', res)
```

Output:

```
First occurrence of 1 is 2
First occurrence of 1 after 4th index is: 6
```

Tuple Membership Test

Utilizing the watchword, we can decide whether a thing is available in the given Tuple.

Code

```
# Python program to show how to perform membership test for tuples
# Creating a tuple
tuple_ = ("Python", "Tuple", "Ordered", "Immutable", "Collection", "Ordered")
# In operator
print('Tuple' in tuple_)
print('Items' in tuple_)
# Not in operator
print('Immutable' not in tuple_)
print('Items' not in tuple_)
```

Output:

```
True
False
False
True
```

Iterating Through a Tuple

A for loop can be used to iterate through each tuple element.

Code

```
# Python program to show how to iterate over tuple elements
# Creating a tuple
tuple_ = ("Python", "Tuple", "Ordered", "Immutable")
# Iterating over tuple elements using a for loop
for item in tuple_:
    print(item)
```

Output:

```
Python
Tuple
Ordered
Immutable
```

Changing a Tuple

Tuples, instead of records, are permanent articles.

This suggests that once the elements of a tuple have been defined, we cannot change them. However, the nested elements can be altered if the element itself is a changeable data type like a list.

Multiple values can be assigned to a tuple through reassignment.

Code

```
# Python program to show that Python tuples are immutable objects
# Creating a tuple
tuple_ = ("Python", "Tuple", "Ordered", "Immutable", [1,2,3,4])
# Trying to change the element at index 2
try:
    tuple_[2] = "Items"
    print(tuple_)
except Exception as e:
    print( e )
# But inside a tuple, we can change elements of a mutable object
tuple_[-1][2] = 10
print(tuple_)
# Changing the whole tuple
tuple_ = ("Python", "Items")
print(tuple_)
```

Output:

```
'tuple' object does not support item assignment
('Python', 'Tuple', 'Ordered', 'Immutable', [1, 2, 10, 4])
('Python', 'Items')
```

The + operator can be used to combine multiple tuples into one. This phenomenon is known as concatenation.

We can also repeat the elements of a tuple a predetermined number of times by using the * operator. This is already demonstrated above.

The aftereffects of the tasks + and * are new tuples.

Code

```python
# Python program to show how to concatenate tuples
# Creating a tuple
tuple_ = ("Python", "Tuple", "Ordered", "Immutable")
# Adding a tuple to the tuple_
print(tuple_ + (4, 5, 6))
```

Output:

```
('Python', 'Tuple', 'Ordered', 'Immutable', 4, 5, 6)
```

Tuples have the following advantages over lists:

- Triples take less time than lists do.
- Due to tuples, the code is protected from accidental modifications. It is desirable to store non-changing information in "tuples" instead of "records" if a program expects it.
- A tuple can be used as a dictionary key if it contains immutable values like strings, numbers, or another tuple. "Lists" cannot be utilized as dictionary keys because they are mutable.

Built-in Functions:

1. **len()**: Returns the number of items in the tuple.

   ```python
   my_tuple = (1, 2, 3, 4, 5)
   print(len(my_tuple))  # Output: 5
   ```

2. **max()**: Returns the largest item in the tuple.

   ```python
   my_tuple = (10, 20, 30, 40, 50)
   print(max(my_tuple))  # Output: 50
   ```

3. **min()**: Returns the smallest item in the tuple.

   ~~~python
   my_tuple = (10, 20, 30, 40, 50)
   print(min(my_tuple))  # Output: 10

4. **sum()**: Returns the sum of all elements in the tuple (only if all elements are numeric).

   ```python
   my_tuple = (1, 2, 3, 4, 5)
   print(sum(my_tuple))  # Output: 15

1. **count()**: Returns the number of occurrences of a specified value in the tuple.

   ```python
   my_tuple = (1, 2, 2, 3, 2, 4, 2)
   print(my_tuple.count(2))  # Output: 4
   ~~~

2. **index()**: Returns the index of the first occurrence of a specified value.

   ```python
   my_tuple = (1, 2, 3, 4, 5)
   print(my_tuple.index(3))  # Output: 2
   ```