# TUPLES

## 1. What is tuple?

- In Python a tuple (pronounced TUH-pul) is an ordered set of values. The separator for each value is often a comma .Common uses for the tuple as a data type are
  - For passing a string of parameters from one program to another, and
  - Representing a set of value attributes in a relational database.
- In some languages, tuples can be nested within other tuples within parentheses or brackets or other delimiters. Tuples can contain a mixture of other data types.
- **A tuple is an immutable sequence of Python objects**. Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

## 2. Defining a Tuple:

To define a tuple, we just have to assign a single variable with multiple values separated by commas, and that variable will be known as a **Tuple**.

***Example:***

```
myTuple = 1, 2, 3, 4
print (myTuple)
```

***Output:***

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

**myTuple** variable is actually a collection of integers 1, 2, 3 and 4. Also, note those **circular brackets** which appears while printing, around the integers, these will actually help you to distinguish between lists and tuples. Because in case of lists, we have **square brackets** around the list elements.

You can obviously **add data of different types** in a single tuple,

```
secondTuple = 1, 2, "python", 4
print secondTuple
```

***Output:***

```
(1, 2, "python", 4)
```

## 3. Accessing Elements In A Tuple:

While working with tuples, we store different data as different tuple elements. Sometimes, there is a need to print a specific information from the tuple. For instance, a piece of code would want just names to be printed of all the fruit data.

### 3.1. Indexing in Tuples:

- Indexing in tuples is also pretty similar to that in lists, the first element has index **zero**, and it keeps on increasing for the next consecutive elements.

- **Backward indexing** is also valid in tuples, i.e., the last element can be accessed using the index -1 and the consecutive previous numbers by -2, -3 and so on.

***Example:***

```
newTuple = "apple", "orange", "banana", "berry", "mango"
newTuple[0]
```

***Output:***

```
'apple'
```

In the table below we have marked the tuple elements for both forward and backward indexing:

| Value | Forward Indexing | Backward Indexing |
|---|---|---|
| **apple** | **0** | **-5** |
| **orange** | **1** | **-4** |
| **banana** | **2** | **-3** |
| **berry** | **3** | **-2** |
| **mango** | **4** | **-1** |

Table 1

### 3.2. Slicing in Tuples:

When you want to extract part of a string, or some part of a list, you use a slice. Slicing in tuples, works exactly the same like in the case of lists.

**Syntax:**

```
string[start:end:step]
```

***Example:***

```
b = 1,2,3,4,5,6,7,8,9
print(b[1:])
print(b[2:4])
print (b[::2])
```

***Output:***

```
(2, 3, 4, 5, 6, 7, 8, 9)
(3, 4)
(1, 3, 5, 7, 9)
```

## 4. Basic Tuples Operations:

- The various operations that we can perform on tuples are very similar to lists.
- Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.
- Some other operators for tuples include:

| Python Expression | Results | Description |
|---|---|---|
| **len((1, 2, 3))** | 3 | Length |
| **(1, 2, 3) + (4, 5, 6)** | (1, 2, 3, 4, 5, 6) | Concatenation |
| **('Hi!',) * 4** | ('Hi!', 'Hi!', 'Hi!', 'Hi!') | Repetition |
| **3 in (1, 2, 3)** | True | Membership |
| **for x in (1, 2, 3): print x,** | 1 2 3 | Iteration |

Table 2

### 4.1 Concatenation of Tuples:

As we know, that tuples are immutable, hence the data stored in a tuple cannot be edited, but it's definitely possible to add more data to a tuple.

This can be done using the **addition operator**.

***Example:***

```
t = (1, 2, 3, 4, 5)
```

In case you want to add another element, 7 to the tuple, then you can do it as follows:

```
t = t + (7,)
```

As you can see, we used the addition operator to add(7,) to the tuple t.

```
print (t)
```

***Output:***

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

#### 4.1.1 Iterate Through Tuple Elements Using For Loop in Python:

You can perform the loop over the tuple elements using the for loop of Python. The for loop iterate each element of the tuple and print the **elements in every single line**. The elements can be of any type like string or integers.

***Example:***

```
s = (5,58,25,17,25,8,2,9)
for i in s:
    print(i)
```

***Output:***

```
5
58
25
17
25
8
2
```

9

### 4.2.2 Loop Through Tuple Elements Using While Loop in Python:

The while loop using the different method from the for loop method.

You have to **first initialize a variable** with zero(0). After that, you have to find the length of tuple for continous iteration.

***Example:***

```
s = (5,58,25)
t=0;
while t < len(s):
      print(s[t])
      t += 1
```

***Output:***

5

58

25

### 4.2. Membership of tuple:

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below

| Operator | Description | Example |
|---|---|---|
| **In** | Evaluates to true if it finds a variable in the specified sequence and false otherwise. | x in y, here in results in a 1 if x is a member of sequence y. |
| **not in** | Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. | x not in y, here not in results in a 1 if x is not a member of sequence y. |

Table 3

***Example:***

```
print (9 in s)
```

***Output:***

```
True
```

***Example:***

```
print (1 in s)
```

***Output:***

```
False
```

## 5. Tuple Methods:

Python has two built-in methods that you can use on tuples.

### 5.1 count( ):

Returns the number of times a specified value occurs in a tuple

***Example:***

```
my_tuple = ('a','p','p','l','e',)
my_tuple.count('p')
```

***Output:***

```
2
```

### 5.2 index( ):

Searches the tuple for a specified value and returns the position of where it was found.

***Example:***

```
my_tuple = ('a','p','p','l','e',)
my_tuple.index('l')
```

***Output:***

```
3
```

**Note: If there is same value multiple times like "p" then it will print the index of 1st item that is being encountered.**

## 6 Change Tuple Values:

- Once a tuple is created, you cannot change its values.
- Tuples are **unchangeable**, or **immutable** as it also is called.
- But there is a workaround.
- We can convert the tuple into a list, change the list, and convert the list back into a tuple.

***Example:***

```
x = ["apple", "banana", "cherry"]
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

***Output:***

```
('apple', 'kiwi', 'cherry')
```

## 7 Deleting a Tuple

- In order to delete a tuple, the del keyword is used.
- In order to delete a tuple named myTuple (which we defined earlier).

***Example:***

```
t = (1, 2, 3, 4, 5)
del myTuple
print (t)
```

***Output:***

```
NameError                                 Traceback (most rece
nt call last)
<ipython-input-16-dea2fd0e44f4> in <module>
----> 1 print (t)

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

***NOTE : WE HAVE SUCESSFULLY DETLETED THE TUPLE NAMED AS t**

# 8 Other examples:

## 8.1 The tuple() Constructor:

- It is also possible to use the tuple() constructor to make a tuple.
- It is use for typecasting or creating tuples .

***Example:***

```
thistuple = tuple(["apple", "banana", "cherry"])
print(thistuple)
```

***Output:***

```
('apple', 'banana', 'cherry')
```

## 8.2 Packing and Unpacking

- In packing, we place value into a new tuple while in unpacking we extract those values back into variables.

***Example:***

```
x = ("Guru99", 20, "Education")    # tuple packing
(company, emp, profile) = x    # tuple unpacking
print(company)
print(emp)
print(profile)
```

***Output:***

```
Guru99
20
Education
```

# 9 Tuples vs List

## 9.1 Syntax Difference

- The literal syntax of tuples is shown by parentheses ( ) whereas the literal syntax of lists is shown by square brackets [ ] .

## 9.2 Mutable Vs. Immutable

- The key difference is that tuples are immutable.
- This means that you cannot change the values in a tuple once you have created it.
- This is a good feature to have in some data structures where you intend to not make any changes to certain parts.

## 9.3 Size Comparison

- Tuples operation has smaller size than that of list, which makes it a bit faster but not that much to mention about until you have a huge number of elements.

## 9.4 Homogeneous Vs Heterogeneous

- There's a strong culture of tuples being for heterogeneous collections, similar to what you'd use structs for in C, and lists being for homogeneous collections, similar to what you'd use arrays for.
- In other words, different data can be stored in single tuple while same type of data is stored in lists.

**Video Link** : https://youtu.be/GstQPTWpt88

**For more detail (optional)** : https://youtu.be/5uDLF9qb-Lk

**References :**

https://www.studytonight.com/python/tuples-in-python
https://www.tutorialspoint.com/python/python_tuples.htm
https://www.geeksforgeeks.org/tuples-in-python/

… Happy Learning ! ...