Tuple:

```
Python tuples are a type of data structure that is very similar to lists. The main difference between the two is that tuples are immutable, meaning they cannot be changed once they are created. This makes them ideal for storing data that should not be modified, such as database records.

A tuple can have any number of items, which may be of different types, such as a string, integer, float, list, etc.
```

Creating a Python Tuple

```
Tuples can be created in a number of ways. The most common way is to wrap them in parentheses. Both single and multiple tuples must always be followed by a comma, like so:
```

```
my_tuple = (item1,) #single tuple
my_tuple = (item1, item2, item3) #multiple tuple
```

In [3]:

```python
t=(1) # if we check the type of data if will give as int
type(t)
```

Out[3]:

```
int
```

```
To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.
```

In [13]:

```python
t=(1,)
type(t)
```

Out[13]:

```
tuple
```

In [7]:

```python
t1 = ("apple", "banana", "cherry")
print(t1)

```

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

Tuple Items

1. Tuple items are ordered:

```
When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.
```

In [8]:
```python
t=(4,1,5,2,6,4,8,9)
t
```

Out[8]:
```
(4, 1, 5, 2, 6, 4, 8, 9)
```

2. Unchangeable:

```
Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.
```

In [9]:
```python
t=("Zubair","Apple",1,2.3,1+2j,)
t
```

Out[9]:
```
('Zubair', 'Apple', 1, 2.3, (1+2j))
```

3. Allow Duplicates:

```
Since tuples are indexed, they can have items with the same value
```

In [10]:
```python
t2 = ("apple", "banana", "cherry",1,2,4,9,True,False, "apple", "cher
print(t2)
```

```
('apple', 'banana', 'cherry', 1, 2, 4, 9, True, False, 'apple', 'cherr
y')
```

```
Tuple Length:
    To determine how many items a tuple has, use the len() function
```

In [12]:
```python
print(len(t2))
```

```
11
```

Python Tuple Methods

```
There are a few methods that can be used on tuples. The most common is the index() method, which can be used to find the position of a given element in the tuple:
```

In [19]:
```python
t = ("a", "b", "c")
t.index("b")
```

Out[19]:
```
1
```

```
Another useful method is count(), which returns the number of times a given element appears in the tuple:
```

In [20]:
```python
t1= ("a", "b", "c", "b")
t1.count("b")

```
Out[20]:
```
2
```

Python Tuple Operations:

1. Accessing Tuples:

```
Tuples can be accessed just like lists, using square brackets and the index of the element you want to access. For example:
```

In [21]:
```python
t1= ("a", "b", "c")
t1[1]
```
Out[21]:
```
'b'
```

In [22]:
```python
t1[2]
```
Out[22]:
```
'c'
```

2. Slicing:

```
You can also access elements from the end of the tuple using negative indices:
```

In [24]:
```python
t4 = ("a", "b", "c")
t4[-3]
```
Out[24]:
```
'a'
```

In [25]:
```python
t4[::-1]
```
Out[25]:
```
('c', 'b', 'a')
```

In [26]:
```python
t4[0:10]
```
Out[26]:
```
('a', 'b', 'c')
```

In [30]:
```python
t4[-3:-1]
```
Out[30]:
```
('a', 'b')
```

In [31]:
```
t5=(1,2,3,4,6,9,2,"zub",True)
t5[0:100:3]
```
Out[31]:
```
(1, 4, 2)
```

3. Joining Tuples:

Tuples can be joined together using the + operator:

In [35]:
```
t1 = ("a", "b")
t2 = ("c", "d")
print(t1 + t2)

```
```
('a', 'b', 'c', 'd')
```

This can also be done with the += operator:

In [37]:
```
t1 = ("a", "b")
t2 = ("c", "d")
t1 += t2  #t1=t1+t2
print(t1)

```
```
('a', 'b', 'c', 'd')
```

In [34]:
```
t1*2
```
Out[34]:
```
('a', 'b', 'a', 'b')
```

Comparing Tuples:

Tuples can be compared with each other using the comparison operators, such as == and !=.
For example:

In [38]:
```
my_tuple = (1, 2)
my_tuple2 = (1, 2)
print(my_tuple == my_tuple2)
```
```
True
```

The comparison is done element by element, so tuples of different lengths can not be equal. Additionally, tuples can be compared with other data types, such as lists:

```
t6 = (1, 2)
l1 = ["a", "b"]
```

```
print(t1 < l1)
```

Nesting Tuples:

```
Tuples can be nested inside each other to create complex data structures.This can be useful for representing data in a more natural way, such as a list of points in a two-dimensional space.
For example:
```

In [40]:
```
t9 = ((1, 2), ("a", "b"))
print(t9)
```

```
((1, 2), ('a', 'b'))
```

Converting Between Tuples and Lists:

```
It's often useful to be able to convert between tuples and lists, as they are similar data types. This can be done using the tuple() and list() functions. For example:
```

In [41]:
```
my_list = ["a", "b", "c"]
my_tuple = tuple(my_list)
print(my_tuple)
```

```
('a', 'b', 'c')
```

```
This can also be done using the built-in zip() function, which takes two or more sequences and returns a list of tuples:
```

In [42]:
```
my_list = ["zub", "apple", "c"]
my_tuple = tuple(zip(my_list))
print(my_tuple)

```

```
(('zub',), ('apple',), ('c',))
```

Python Tuples vs Lists

```
As we've seen, tuples and lists are very similar data structures.
The main difference is that tuples are immutable, while lists are mutable. This means that tuples cannot be changed once they are created, while lists can be modified after they are created.
This also means that tuples can be used as keys in dictionaries, while lists cannot.
Another difference is that tuples are typically faster than lists. This is because Python knows that a tuple cannot be changed, so it doesn't need to allocate as much memory for it.
```