TUPLES

A tuple is an immutable list, meaning that once it is defined it cannot be changed. This is different from normal lists in which you can add, remove, and change the values. With tuples none of these actions are an option. Where tuples are similar to lists is that they are ordered in the same fashion, and the values in the tuple can be still be accessed by index. In fact, you can perform many of the same operations on a tuple that you can on a list. You can concatenate tuples, you can iterate over the values in a tuple with a **for** loop, you can access values from the end of the tuple using negative indices, and you can access slices of a tuple.

Tuples are created using comma separated values between parentheses. The pattern is **tuple_name = (item_1, item_2, item_N)**. If you only want a single item in a tuple, that single item must always be followed by a comma. The pattern is **tuple_name = (item_1,)**.

 # Creating a Tuple

```
>>> days_of_week = ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday')

>>> days_of_week

('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday')


# You cannot modify values in a tuple. This will raise an exception.
>>> days_of_week[0] = 'New Monday'

Traceback (most recent call last):

  File "<pyshell#4>", line 1, in <module>

    days_of_week[0] = 'New Monday'

TypeError: 'tuple' object does not support item assignment

>>>

>>>

>>> # creating tuple of strings

>>> my_tuple1 = ("hi", "hello")
```

```
>>> my_tuple1
('hi', 'hello')
>>>
>>> my_tuple2 = (1, 3.5, "hi")   # creating a tuple with mixed datatypes
>>> my_tuple2
(1, 3.5, 'hi')
>>>
>>> my_tuple3 = ("Welcome", [1,2,3])   # creating a tuple with string and list objects
>>> my_tuple3
('Welcome', [1, 2, 3])
>>>
>>> my_tuple4 = ((1,2,3), (4,5,"hello"))   # nested tuples
>>> my_tuple4
((1, 2, 3), (4, 5, 'hello'))
>>>
>>> my_tuple5 = ()    # creating an empty tuple
>>> my_tuple5
()
>>> my_tuple6 = (10, )   # creating a tuple with single element
>>> my_tuple6
(10,)
>>>
# Accessing (indexing) tuple elements
>>> my_tuple1[0]
'hi'
```

```
>>> my_tuple2[2]
'hi'
>>> my_tuple3[1]
[1, 2, 3]
>>>
>>> my_tuple4[1][1]
5
>>>
```

Note: It is not possible to delete an element of a tuple because of its immutable nature. But you can delete an entire tuple

```
>>> my_tuple4
((1, 2, 3), (4, 5, 'hello'))
>>> del my_tuple4[0]
Traceback (most recent call last):
  File "<pyshell#38>", line 1, in <module>
    del my_tuple4[0]
TypeError: 'tuple' object doesn't support item deletion
>>>
>>> del my_tuple4
>>> my_tuple4


Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    my_tuple4
NameError: name 'my_tuple4' is not defined
>>>
```

>>> **# membership test in tuples**

Membership tests in tuple consists of 'In' and 'Not in' members. While "in" checks whether a element exists in a particular tuple and "not in" checks if a element does not exist in a tuple.

```
>>> my_tuple7 = (11,22,33,44,55,66,77,88,99)
>>> print(22 in my_tuple7)
True
>>> print(75 in my_tuple7)
False
>>> print(100 not in my_tuple7)
True
>>>
>>>
>>> # Iterating a tuple
>>>
>>> my_tuple8 = ("Apple", "Mango", "Banana", "Orange")
>>> for fruits in my_tuple8:
        print(fruits)


Apple
Mango
Banana
Orange
>>>
```

 # Switching between Tuples and Lists

In order to convert a data to a tuple object, we use the **tuple()** function. Similarly, to convert a tuple to a list object, we use the **list()** function.

```
>>> animals_list = ['Lion', 'Tiger', 'Bull']

>>> animals_list

['Lion', 'Tiger', 'Bull']

>>> animals_tuple = tuple(animals_list)  #converting a list object to tuple

>>> animals_tuple

('Lion', 'Tiger', 'Bull')

>>>

>>> my_list = list(my_tuple12)   # converting a tuple object to a list

>>> my_list

[(1, 2, 3), ('hello', 'Python')]

>>>
```

```
>>> # Tuple Unpacking / Tuple Assignment
```

You can use tuples to assign values to multiple variables at the same time.

```
>>> mon = days_of_week[0]

>>> print(mon)

Monday

>>>

>>> my_tuple9 = (1,2,3,4,5)

>>> x,y,z,a,b = my_tuple9

>>> x

1
```

```
>>> a
4
>>>
>>>
```

 # Concatenating Tuples

```
>>> my_tuple10 = (1,2,3)
>>> my_tuple11 = ("hello", "Python")
>>> print(my_tuple10 + my_tuple11)
(1, 2, 3, 'hello', 'Python')
>>>
```

>>> **# Nesting Tuples**

```
>>> my_tuple12 = (my_tuple10, my_tuple11)
>>> my_tuple12
((1, 2, 3), ('hello', 'Python'))
>>>
>>>
```

 # Tuple methods

Index(): displays the index position of a particular tuple element

Count(): displays the number of occurrences of a particular tuple element.

```
>>> my_tuple13 = (1,2,3,1)
>>> my_tuple13.count(1)
2
>>> my_tuple13.index(2)
1
```