TUPLE OPERATION IN PYTHON

Tuple

- A tuple is a sequence of values.
- A tuple in Python is similar to a list.
- The important difference is that tuples are immutable. (we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list).

Tuple creation

- A tuple is created by placing all the items (elements) inside parentheses (), separated by commas.
- The **parentheses are optional**, however, it is a good practice to use them.
- A tuple can have any number of items and they may be of different types (integer, float, list, string, etc.).
- Another way to create a tuple is the built-in function tuple. With no argument, it creates an empty tuple:

```
>>> t = tuple()
>>> print t
()
```

```python
# Different types of tuples

# Empty tuple
my_tuple = ()
print(my_tuple)

# Tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)

# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple)

# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)
```

Output

```
()
(1, 2, 3)
(1, 'Hello', 3.4)
('mouse', [8, 4, 6], (1, 2, 3))
```

Tuple creation

- A tuple can also be created without using parentheses. This is known as tuple packing.

```
my_tuple = 3, 4.6, "dog"
print(my_tuple)

# tuple unpacking is also possible
a, b, c = my_tuple

print(a)      # 3
print(b)      # 4.6
print(c)      # dog
```

Output

```
(3, 4.6, 'dog')
3
4.6
dog
```

If swap values of a and b:
>>> a, b = b, a

Tuple creation

- Creating a tuple with one element is a bit tricky.
- Having one element within parentheses is not enough. We will need a trailing comma to indicate that it is, in fact, a tuple.

```
my_tuple = ("hello")
print(type(my_tuple))  # <class 'str'>

# Creating a tuple having one element
my_tuple = ("hello",)
print(type(my_tuple))  # <class 'tuple'>

# Parentheses is optional
my_tuple = "hello",
print(type(my_tuple))  # <class 'tuple'>
```

Output

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

Tuples as return values

Example:

built-in function divmod takes two arguments and returns a tuple of two values, the quotient and remainder.

```
>>> t = divmod(7, 3)
>>> print t
(2, 1)
```

- Or use tuple assignment to store the elements separately:

```
>>> quot, rem = divmod(7, 3)
>>> print quot
2
>>> print rem
1
```

Access Tuple Elements

- There are various ways in which we can access the elements of a tuple.

1. **Indexing**
2. **Negative Indexing**
3. **Slicing**

Indexing

```
# Accessing tuple elements using indexing
my_tuple = ('p','e','r','m','i','t')

print(my_tuple[0])   # 'p'
print(my_tuple[5])   # 't'

# IndexError: list index out of range
# print(my_tuple[6])

# Index must be an integer
# TypeError: list indices must be integers, not float
# my_tuple[2.0]

# nested tuple
n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

# nested index
print(n_tuple[0][3])       # 's'
print(n_tuple[1][1])       # 4
```

Output

```
p
t
s
4
```

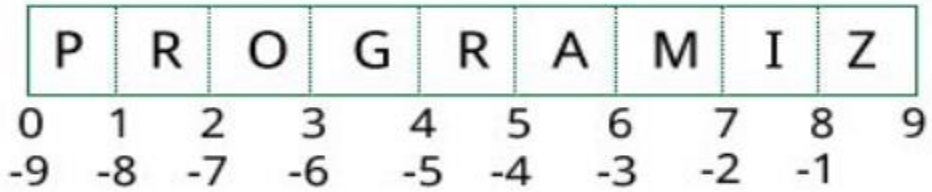
Negative Indexing

```
# Negative indexing for accessing tuple elements
my_tuple = ('p', 'e', 'r', 'm', 'i', 't')

# Output: 't'
print(my_tuple[-1])

# Output: 'p'
print(my_tuple[-6])
```

Output

```
t
p
```

Slicing




```
# Accessing tuple elements using slicing
my_tuple = ('p','r','o','g','r','a','m','i','z')

# elements 2nd to 4th
# Output: ('r', 'o', 'g')
print(my_tuple[1:4])

# elements beginning to 2nd
# Output: ('p', 'r')
print(my_tuple[:-7])

# elements 8th to end
# Output: ('i', 'z')
print(my_tuple[7:])

# elements beginning to end
# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
print(my_tuple[:])
```

Output

```
('r', 'o', 'g')
('p', 'r')
('i', 'z')
('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
```

Immutable

```
# Changing tuple values
my_tuple = (4, 2, 3, [6, 5])


# TypeError: 'tuple' object does not support item assignment
# my_tuple[1] = 9

# However, item of mutable element can be changed
my_tuple[3][0] = 9    # Output: (4, 2, 3, [9, 5])
print(my_tuple)

# Tuples can be reassigned
my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
print(my_tuple)
```

```
# Concatenation
# Output: (1, 2, 3, 4, 5, 6)
print((1, 2, 3) + (4, 5, 6))

# Repeat
# Output: ('Repeat', 'Repeat', 'Repeat')
print(("Repeat",) * 3)
```

Deleting a Tuple

- we cannot delete or remove items from a tuple.
- Deleting a tuple entirely, however, is possible using the keyword del.

```
# Can delete an entire tuple
del my_tuple
```

Tuple Methods

- Methods that add items or remove items are not available with tuple. Only the following two methods are available.
- Some examples of Python tuple methods:

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

print(my_tuple.count('p'))  # Output: 2
print(my_tuple.index('l'))  # Output: 3
```

Other Tuple Operations

Tuple Membership Test

```
# Membership test in tuple
my_tuple = ('a', 'p', 'p', 'l', 'e',)

# In operation
print('a' in my_tuple)
print('b' in my_tuple)

# Not in operation
print('g' not in my_tuple)
```

Output

```
True
False
True
```

Iterating Through a Tuple

```
# Using a for loop to iterate through a tuple
for name in ('John', 'Kate'):
    print("Hello", name)
```

```
Hello John
Hello Kate
```

Advantages of Tuple over List

- We generally use tuples for **heterogeneous (different)** data types and lists for **homogeneous (similar)** data types.
- Since tuples are **immutable,** iterating through a tuple is **faster** than with list. So there is a slight performance boost.
- Tuples that contain immutable elements can be used as a **key for a dictionary**. With lists, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains **write-protected.**

Summarize

- List
- Dictionary
- Set
- Tuple

Summary-Comparison chart

Lists	Tuples	Sets	Dictionaries
A list is a collection of *ordered* data.	A tuple is an *ordered* collection of data.	A set is an *unordered* collection.	A dictionary is an *unordered* collectio n of data that stores data in key-value pairs.
Lists are *mutable*.	Tuples are *immutable*.	Sets are *mutable* and have *no duplicate elements*.	Dictionaries are mutable and keys do not allow duplicates.
We can represent a List by [ ]	We can represent a Tuple by ( )	We can represent a Set by { }	We can represent a Dictionary by { }
It is ordered in nature.	It is ordered in nature.	It is unordered in nature.	It is ordered in nature.