In Python programming, tuple is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas in a list, elements can be changed.

Advantages of Tuple over List

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

Creating a Tuple

A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma. The parentheses are optional but is a good practice to write it. A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).

Example

```
my_tuple = (1,3.2,"mouse", [8, 4, 6], (1, 2, 3))

print(my_tuple)
```

Accessing Elements in a Tuple

We can use the index operator [] to access an item in a tuple where the index starts from 0. So, a tuple having 6 elements will have index from 0 to 5. Trying to access an element other that (6, 7,...) will raise an IndexError. The index must be an integer, so we cannot use float or other types. This will result into TypeError. Likewise, nested tuple are accessed using nested indexing.as shown in the example below.Python allows negative indexing for its sequences.

Example

```
n_tuple = (1,"mouse", [8, 4, 6], (1, 2, 3))

# Output: 's'

print(n_tuple[1][3])

# Output: 4

print(n_tuple[2][1])

# Output: 1

print(n_tuple[0])

# Output: [8, 4, 6]

print(n_tuple[-2])
```

Slicing

We can access a range of items in a tuple by using the slicing operator - colon ":".
Example

```
my_tuple = ('p','r','o','g','r','a','m')

# Output: ('r', 'o', 'g')

print(my_tuple[1:4])

# Output: ('p', 'r')

print(my_tuple[:-5])

# Output: ('a', 'm')

print(my_tuple[5:])

# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm')

print(my_tuple[:])

#('m', 'a', 'r', 'g', 'o', 'r', 'p')

print my_tuple[::-1]

#('r', 'p')

print my_tuple[1::-1]
```

Changing a Tuple

Unlike lists, tuples are immutable.This means that elements of a tuple cannot be changed once it has been assigned. But, if the element is itself a mutable datatype like list, its nested items can be changed. We can also assign a tuple to different values (reassignment).

Example

```
my_tuple = (4, 2, 3, [6, 5])

my_tuple[3][0] = 9

print(my_tuple)#(4, 2, 3, [9, 5])

my_tuple[1]=40#TypeError: 'tuple' object does not support item assignment

print(my_tuple)
```

We can use + operator to combine two tuples. This is also called **concatenation**.

We can also **repeat** the elements in a tuple for a given number of times using the * operator.

Both + and * operations result into a new tuple.
Example

```
# Concatenation

# Output: (1, 2, 3, 4, 5, 6)

print((1, 2, 3) + (4, 5, 6))

# Repeat

# Output: ('Repeat', 'Repeat', 'Repeat')

print(("Repeat",) * 3)
```

Edit the tuple using '+' and slice(:) operators

We can peform all the operations like append the element/elements at the end of a tuple, insert one element/elements in one particular position , remove and replce the values using '+' operator and ':'(slice) operator.

Example

```
mytuple=input("Enter the elements to the tuple")

print mytuple

print mytuple#1,3,3,3,5,6

mytuple=mytuple[:1]+input("Enter a number")+mytuple[1:]#insert the value in one place , ex:2,

print mytuple#(1, 2, 3, 3, 3, 5, 6)

mytuple=mytuple[:3]+input("Enter a number")+mytuple[4:]#replace the value with another value

print mytuple#(1, 2, 3, 4, 3, 5, 6)

mytuple=mytuple[:4] + mytuple[5:]#Remove one element

print mytuple#(1, 2, 3, 4, 5, 6)

my_tuple=reversed(mytuple)#it will return reversed object

print tuple(my_tuple)
```

Deleting a Tuple

we cannot delete or remove items from a tuple.But deleting a tuple entirely is possible using the keyword del.

Example

my_tuple = ('p','r','o','g','r','a','m')

del my_tuple

Tuple Methods

Methods that add items or remove items are not available with tuple. Only the following two methods are available.

index() method

The index method takes a single argument (element) and finds the given element in a tuple and returns its position.However, if the same element is present more than once, index() method returns its first position.

Syntax

tuple.index(element)

count() method

The count() method takes a single argument(element) and counts how many times an element has occurred in a tuple and returns it.

Syntax

tuple.count(element)

Example

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

print(my_tuple.count('p'))#Output:2

print(my_tuple.index('l'))#Output:3

Tuple functions

all(): It takes any iterable as a parameter and return True if all elements of the tuple are true(tuple should not contains any 0 value or False value) or if the tuple is empty.

Syntax : all(iterable)

any(): It takes any iterable as a parameter and Return True if any element of the tuple is true. If the tuple is empty, return False.

Syntax: any(iterable)

len() : It takes the sequence as a parameter and return the length (the number of items) in the tuple.

Syntax: len(sequence)

sum(): Return the sum of all elements in the tuple.Normally, items of the iterable should be numbers. The sum() function adds *start* and items of the given *iterable* from left to right.This start is optional. The default value of *start* is 0 (if omitted)

Syntax: sum(iterable, start)

tuple(): It takes the sequence as an parameter and Convert an iterable (list, string, set, dictionary) to a tuple. If no parameters are passed, it creates an empty tuple

Syntax: tuple([iterable])

max() : The method **max** returns the elements from the *tuple* with maximum value.While giving an iterable as an argument we must make sure that all the elements in the iterable are of the same type. If we pass different type of element to a tuple it fallows the priority to return the maximum value. **Order of priority** :Tuple,string,Char,list,number.

Syntax:max(tuple)

Example

```
print max((2,3))

print max(('A','b','c','B'))

print max(("cat","bird","apple","cb"))#based on the ascii value of first letter

print max(([1,2,3],[3,1,2],[3,3,2]))#comparing the first element of each list
```

min() :The method **min()** returns the elements from the *tuple* with minimum value.While giving an iterable as an argument we must make sure that all the elements in the iterable are of the same type. If we pass different type of element to a tuple it fallows the priority to return the minimum value. **Order of priority** : number,list,char,string,tuple

Syntax:min(tuple)

sorted() : The sorted() method sorts the elements of a given iterable in a specific order - Ascending or Descending. Return a new sorted list(does not sort the tuple itself).

Syntax

sorted(tuple)

To print the values in Descending order Syntax is

sorted(tuple,reverse=True)

enumerate()

The enumerate() method adds counter to an iterable and returns it. The returned object is a enumerate object. It contains the index and value of all the items of list as a tuple.

Syntax

enumerate(iterable, start=0)

reversed()

The reversed() function returns an iterator object that accesses the given sequence in the reverse order.we have to convert that iterator object into tuple by using tuple function.

Syntax

reversed(seq)

Example

```
pyTuple =(20,55,43,22,67,90,0)

print all(pyTuple)

print any(pyTuple)

print len(pyTuple)

print max(pyTuple)

print min(pyTuple)

print sum(pyTuple)

print sorted(pyTuple)

a=enumerate(pyTuple)

print tuple(a)

for item in enumerate(pyTuple):

  print(item)

mytuple=reversed(pyTuple)

print tuple(mytuple)
```

Exercise:

1.Write a Python program to count the elements in a tuple and add the elements to a new tuple until an element is a tuple.Print the sum of nested tuple in the tuple?

2. Write a Python program to replace last value of tuples in a list with A,B,C grades?

 Example:mytuple=((1,2,70),(3,4,80),(5,6,90)

Output: ((1,2,'C'),(3,4,'B'),(5,6,'A'))

3. Write a Python program to check whether an element exists within a tuple or not without using any methods and functions?