TUPLES

- Similar to a list
- Immutable
- We cannot perform operations like append(), extend(), insert(), remove(), pop() and clear() on tuples.
- Used to store data which should not be modified and retrieve that data on demand.

Creating a Tuple

Example:

```
tuple1 = () #empty tuple
print(tuple)
tuple() #empty tuple
print(tuple)
tuple2 = (1,) #single element tuple
print(tuple)
tuple3=(1,2,3) #tuple having integers
tuple4=(1,"Hello",-3.4,50.8) #tuples with mixed datatypes

tuple = 1, 2,3 #no braces (Tuple Packing)
tuple= ("hello",[4,5,6],(1,2,3)) #nested tuple

#create a tuple by using list
list=[1,2,3]
tuple1=tuple(list) #convert list into tuple
print(tuple1)

#creating the tuple using range function
tuple2=tuple(range(1,10,1))
print(tuple2)
```

Example of Tuple Unpacking:

```
tuple1=3, 4.6, "Good Morning"  #tuple Packing

print(tuple1)

a,b,c = tuple1 #tuple unpacking is also possible

print(a)

print(b)

print(c)
```

Example:

```
tuple1=("hello")
print(type(tuple1))
tuple1=("hello",)
print(type(tuple1))
tuple1="hello"
print(type(tuple1))
tuple1="hello" ,
print(type(tuple1))
```

ACCESSING TUPLES

1. **Indexing**

- Index operator []
- Index Error
- Type Error
- Nested indexing

Example:

```
tuple=('i','n','d','e','x','i','n','g')
print(tuple[0])
print(tuple[6])
print(tuple[9]) #index Error
print(tuple[4.0]) #Type Error


#nested tuple
tuple1=("hello",[4,5,6],(1,2,3))
print(tuple1 [0][3]) #nested indexing
print(tuple1 [1][1])
print(tuple1 [2][1])
```

2. **Slicing**

- Slicing operator (:)

  Example:

```
tuple1=('p','r','o','g','r','a','m')
print(tuple1[1:4])
print(tuple1[:-5])
print(tuple1[:]) #beg to end
```

WORKING WITH TUPLES

1. Changing a Tuple

- Elements of the tuple cannot be changed.
- Reassignment is possible.

  Example:

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

#tuple[1]=9 #Type Error

tuple [3][0]=9
print(tuple)

#Tuples can be reassigned
tuple=('H','e','l','l','o')
print(tuple)
```

2. Deleting a Tuple

- We cannot change the elements of the tuple.
- We can delete the entire tuple using **del** keyword.

  Example:

```
tuple1=('p','r','o','g','r','a','m')
del tuple1[3] #type Error
del tuple1 #del the entire tuple
print(tuple1) #Name Error
```

OPERATIONS ON TUPLES

1. **Finding Length**
2. **Concatenation**

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

3. **Repetition**

   Example: print(("hello")*3)

4. **Membership**

   Example:

   ```
   tuple1=('p','r','o','g','r','a','m')
   print('p' in tuple1)
   print('s' in tuple1)
   print('g' not in tuple1)
   ```

5. **Iterations Operation**

   Example:

   ```
   #Iterating through a tuple for loop
   for name in ('Khushi','Payal','Vivek'):
       print("Hello", name)
   ```

FUNCTIONS

- Len()
- Min()
- Max()
- Sorted()

METHODS

- **Count()** – retunes how many time element ‘x’ is found in tuple.
- **Index() –** returns the first occurrences of the element ‘x’ in the tuple. Raises **‘Value Error’** if ‘x’ is not found in the tuple.

Example:

```
tuple=('h','e','l','l','o')
print(tuple.count('l'))
print(tuple.count('o'))
print(tuple.index('o'))
```

Nesting Tuples

- Tuple inserted inside another tuple.

  Example:

  ```
  tup=(10,20,30,40,50,(100,200)) #tuple with 6 elements
  print("Nested tuple=",tup[5])
  ```

Sorting Nested Tuples

- Sorted() function

  Example:

  ```
  emp=((1,"Ram",1000),(22,"Pooja",2000),(13,"Abhishek",5000),(4,"Vivek",20000.50))
  print(sorted(emp))
  ```

Advantages of Tuple Over List

- Use Tuples for heterogeneous data types and lists for homogeneous data types.
- Tuples are immutable, iterating through a tuple is faster than with list.
- Tuple that contain immutable elements can be used as 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.