Sets

A set is an unordered collection of elements much like a set in mathematics. The order of elements is not maintained in the sets. It means the elements may not appear in the same order as they are entered into the set. Moreover a set does not accept duplicate elements. There are two subtypes in sets.

1) Set datatype 2) Frozenset datatype

Set Datatype

To create a set, we should enter the elements separated by com.mas inside curly braces{}.

In [6]:

```
s = {10, 20, 30, 20, 50}
print(s)# may display {50, 10, 20, 30}
```

```
{10, 20, 50, 30}
```

Please observe that the set 's' is not maintaining the order of the elements. We entered the elements in the order 10, 20, 30, 20 and 50. But it is showing another order. Also, we repeated the element 20 in the set, but it has stored only one 20. We can use the setQ function to create a set as:

In [33]:

```
ch = set("Hello")
print(ch)# may display {'H' , 'e', 'L' ,  'o'}

lst = [1,2,5,4,3]
s = set(lst)
print(s)# may display {1, 2, 3, 4, 5}                        "
```

```
{'o', 'e', 'H', 'l'}
{1, 2, 3, 4, 5}
```

Since sets are unordered, we cannot retrieve the elements using indexing or slicing operations. For example, the following statements will give error messages:

In [20]:

```
print(s[0])   #indexing, display oth element
print(s[0:2]) #slicing, display from 0 to 1st elements
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-b85e1a828293> in <module>
----> 1 print(s[0])   #indexing, display oth element
      2 print(s[0:2]) #slicing, display from 0 to 1st elements

TypeError: 'set' object is not subscriptable
```

In [ ]:

```
The update() method is used to add elements to a set as:
```

In [ ]:

```
s.update([50,60])
print(s) #may display {1, 2, 3, 4, 5, 50, 60}
```

In [ ]:

```
On the other hand, the remove() method is used to remove any particular element from a set
```

In [ ]:

```
s.remove(50)
print(s) #may display {1, 2, 3, 4, 5, 60}
```

Frozenset Datatype

The frozenset datatype is same as the set datatype. The main difference is that the elements in the set datatype can be modified; whereas, the elements of frozenset cannot be mod ified. We can create a frozenset by passing a set to frozenset() function as:

In [23]:

```
s = {50,60,70,80,90}
print(s) #may d1splay {80, 90, 50, 60, 70}
```

```
{70, 80, 50, 90, 60}
```

In [37]:

```
fs = frozenset(s)  #create frozenset fs
print(fs)          # may display frozenset({80, 90, SO, 60, 70})
```

```
frozenset({1, 2, 3, 4, 5})
```

Another way of creating a frozenset is by passing a string (a group of characters) to the frozenset() function as:

In [24]:

```
fs = frozenset("abcdefg")
print(fs)  # may display frozenset({'e', 'g', 'f', 'd' , 'a', 'c', 'b'})
```

```
frozenset({'a', 'f', 'b', 'c', 'g', 'e', 'd'})
```

However, update() and remove() methods will not work on frozensets since they can't be modified or updated.

Tuple Datatype

A tuple is similar to a list. A tuple contains a group of elements which can be of different types. The elements in the tuple are separated by commas and enclosed in parentheses (). Whereas the list elements can be modified, it is not possible to modify the tuple elements. That means a tup be treated as a read-only list.

In [ ]:

```
tpl  = (10 , -20 , 15.5, 'vijay', "Mary")
```

The individual elements of the tuple can be referenced using square braces as tpl[0], tpl[1], tpl[2],.... Now, if we try to modify the 0th element as:

In [4]:

```
tpl[0]  = 99
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-787169012ee6> in <module>
----> 1 tpl[0]  = 99

NameError: name 'tpl' is not defined
```

Tuples are immutable which means you cannot update or change the values of tuple elements. This will result in error.

The slicing operations which can be done on lists are also valid in tuples.

In [41]:

```
tpl  = (10 , -20 , 15.5, 'vijay', "Mary")
print(tpl)
```

```
(10, -20, 15.5, 'vijay', 'Mary')
```

In [43]:

```
print(tpl[0])
```

```
10
```

In [44]:

```
print (tpl[1:3])
```

```
(-20, 15.5)
```

In [45]:

```
print (tpl[-2])
```

```
vijay
```

In [46]:

```
print(tpl*2)
```

```
(10, -20, 15.5, 'vijay', 'Mary', 10, -20, 15.5, 'vijay', 'Mary')
```

In [5]:

```
tpl[0]=99
# this will give error
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-cbf94f962571> in <module>
----> 1 tpl[0]=99
      2 # this will give error

NameError: name 'tpl' is not defined
```

Inserting the element 5 in a tuple

In [50]:

```
a=range(10)
tup = tuple(a)
tup
```

Out[50]:

```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

In [52]:

```
tup1 = tup[0:5]
tup1
```

Out[52]:

```
(0, 1, 2, 3, 4)
```

In [53]:

```
tup2 = tup[6: ]
tup2
```

Out[53]:

```
(6, 7, 8, 9)
```

In [55]:

```
tup = tup1 + tup2
tup
```

Out[55]:

```
(0, 1, 2, 3, 4, 6, 7, 8, 9)
```

Adding a new element 20 to a tuple

In [56]:

```
tup
```

Out[56]:

```
(0, 1, 2, 3, 4, 6, 7, 8, 9)
```

In [58]:

```
tup1 = tup[0:5]
tup2 = tup[5: ]

tup_new = tup1 + (20,) + tup2
tup_new
```

Out[58]:

```
(0, 1, 2, 3, 4, 20, 6, 7, 8, 9)
```