Touples:

- they are immutable
- they can hold multiple objects and are like that of the lists
- use of touples is that data is write protected ie cannot be modified
- touples are faster than lists
- touples help in returning multiple values from a function
- they can be heterogenous

note: list and dictionary are mutable tuple and strings are immutable

note: list and tuple can be added but dictionary cannot added

In [2]:

```
# defining a touple:

t=(1,3,"cbh",32.45,45+3j)
print(type(t))

print(t[2])
```

```
<class 'tuple'>
cbh
```

In [8]:

```
# convert touple into a list:
# after conversion of touple into a list it becomes mutable.

print(t)
l=list(t)
print(l)

l[0]=4
print(l)
```

```
(1, 3, 'cbh', 32.45, (45+3j))
[1, 3, 'cbh', 32.45, (45+3j)]
[4, 3, 'cbh', 32.45, (45+3j)]
```

In [11]:

```
#convert list to a touple:

r=tuple(l)
print(r)
print(l)
```

```
(4, 3, 'cbh', 32.45, (45+3j))
[4, 3, 'cbh', 32.45, (45+3j)]
```

In [20]:

```
# concatenation: possible with tuple and list but not with dictionary

print(t)
print(r)
a=t+r
print(a)
print(type(a))
```

```
(1, 3, 'cbh', 32.45, (45+3j))
(4, 3, 'cbh', 32.45, (45+3j))
```

```
(1, 3, 'cbh', 32.45, (45+3j), 4, 3, 'cbh', 32.45, (45+3j))
<class 'tuple'>
```

In [23]:

```
# repetition:

print(t)
a=t*3
print(a)
```

```
(1, 3, 'cbh', 32.45, (45+3j))
(1, 3, 'cbh', 32.45, (45+3j), 1, 3, 'cbh', 32.45, (45+3j), 1, 3, 'cbh', 32.45, (45+3j))
```

In [24]:

```
# max and min of a number

t=(1,2,4,5,67)
print(max(t))
print(min(t))
```

```
67
1
```

In [26]:

```

```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-26-943cdf19ab94> in <module>()
      1 # delete a tuple:
      2
----> 3 print(t)
      4 del(t)

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

sets

- cannot contain same elements, if present they are discarded.
- You cannot access items in a set by referring to an index, since sets are unordered the items has no index.
- elements are not in the order which we gave and not also in a sorted order, they are in an order which is best for internal implementation

In [27]:

```
s={1,4,3,5,5,2,6,1}
print(type(s))
print(s)
```

```
<class 'set'>
{1, 2, 3, 4, 5, 6}
```

In [47]:

```
a=set([1,32,4,34])
print(a)
print(type(a))
```

```
{32, 1, 34, 4}
<class 'set'>
```

In [36]:

```
#adding new element:

print(a)
a.add(443)
print(a)
```

```
{32, 1, 34, 4}
{32, 1, 34, 4, 443}
```

In [40]:

```
#union of sets:

print(a)
print(s)
print(a.union(s))

#OR
print(a|s)
```

```
{32, 1, 34, 4, 443}
{1, 2, 3, 4, 5, 6}
{32, 1, 34, 2, 4, 3, 5, 6, 443}
{32, 1, 34, 2, 4, 3, 5, 6, 443}
```

In [42]:

```
#intersection of sets:

print(a)
print(s)
print(a.intersection(s))
print(a&s)
```

```
{32, 1, 34, 4, 443}
{1, 2, 3, 4, 5, 6}
{1, 4}
{1, 4}
```

In [45]:

```
# difference operator

print(a-s)
print(s.difference(s))
print(a.difference(s))
```

```
{32, 34, 443}
set()
{32, 34, 443}
```

In [48]:

```
# clearing a set:

print(a)
a.clear()
print(a)
```

```
{32, 1, 34, 4}
set()
```

In [49]:

```
# finding an element:

if 4 in s:
    print("present")
else:
```

```
    print("absent")
```

present

In [52]:

```
# xor operator:(give those which are only in one of the sets)

s1={"23",42}
s2={"hello",234,54,42,"42"}
s1^s2
```

Out[52]:

{'23', 234, '42', 54, 'hello'}