TUPLE

In [ ]:
```
# Tuple is an in-built DataStructure
# Tuple is collection of items enclosed/defined with parenthesis() and separated
# Tuple is an ordered collection of item with positional index(i)
# Duplicates are allowed in to Tuple
# Tuple is IMMUTABLE datastructure which means it cannot changed once it is crea
# Tuple allows mutliple datatypes in to it
# Both tuple and list similar but there is a major difference is tuple is immuta
# Tuple consists only two methods count(),index()
```

CREATING A TUPLE

In [9]:
```
# Tuple is defined using parenthesis()
Empty = ()
Empty
```

Out[9]:
```
()
```

In [10]:
```
type(Empty)
```

Out[10]:
```
tuple
```

In [11]:
```
empty = tuple()
empty
```

Out[11]:
```
()
```

In [12]:
```
type(empty)
```

Out[12]:
```
tuple
```

In [13]:
```
# TUPLE OF INTEGERS
t = (1,2,3,4)
t
```

Out[13]:
```
(1, 2, 3, 4)
```

In [14]:
```
# TUPLE OF FLOAT NUMBERS
t1 = (1.2,1.3,1.4,1.5)
t1
```

Out[14]:
```
(1.2, 1.3, 1.4, 1.5)
```

In [15]:
```
# TUPLE OF STRING
t2 = ('hi','welcome','to','python','world')
t2
```

Out[15]:
```
('hi', 'welcome', 'to', 'python', 'world')
```

In [18]:
```
# if we work with single string we need to use TRAILRING COMMA other wise it tre
tt = ("hello")
```

```
tt
type(tt)
```

Out[18]: str

In [19]:
```
tt = ("hello",)
tt
type(tt)
```

Out[19]: tuple

In [20]:
```
tt1 = (1)# not only for string for every single datatype we must use trailing co
tt1
type(tt1)
```

Out[20]: int

In [21]:
```
tt1 = (1,)# not only for string for every single datatype we must use trailing c
tt1
type(tt1)
```

Out[21]: tuple

In [16]:
```
# TUPLE OF COMPLEX
t3 = (1+2j,1-2j)
t3
```

Out[16]: ((1+2j), (1-2j))

In [17]:
```
# TUPLE OF BOOLEAN
t4 = (1,True,0,False)
t4
```

Out[17]: (1, True, 0, False)

In [22]:
```
# TUPLE OF LIST
t5 = ([1,2],[2,3],[3,4])
t5
```

Out[22]: ([1, 2], [2, 3], [3, 4])

In [23]:
```
# TUPLE OF SET
t6 = ({1,2},{3,4})
t6
```

Out[23]: ({1, 2}, {3, 4})

In [24]:
```
# TUPLE OF DICTIONARY
t7 = ({'key':'value'})
t7
type(t7)
```

Out[24]: dict

In [26]:
```
t7 = ({'key':'value'},)
t7
type(t7)
```

Out[26]: tuple

METHODS IN TUPLE

In [ ]:
```
# THERE ARE ONLY TWO METHODS IN TUPLE
# COUNT()....>return no.of times a single item occured in tuple
# INDEX()....>returns index of given value....>if similar item occured multiple
```

count()

In [28]:
```
t8 = (1,2,3,4,2,3,1,4,5,2,3,5,6,7,8,9,10)
t8.count(2)
```

Out[28]: 3

In [29]:
```
t8.count(3)
```

Out[29]: 3

In [30]:
```
t8.count(4)
```

Out[30]: 2

In [31]:
```
t8.count(5)
```

Out[31]: 2

In [32]:
```
t8.count(1)
```

Out[32]: 2

index()

In [34]:
```
t8.index(4)
```

Out[34]: 3

In [35]:
```
t8.index(2)
```

Out[35]: 1

In [39]:
```
t8.index(1)
```

Out[39]: 0

In [37]:
```
t8.index(4)
```

Out[37]: 3

In [38]:
```
t8.index(5)
```

Out[38]: 8

Built-In Functions

In [40]:
```
max(t8)
```

Out[40]: 10

In [41]:
```
min(t8)
```

Out[41]: 1

In [42]:
```
sum(t8)
```

Out[42]: 75

In [43]:
```
len(t8)
```

Out[43]: 17

In [44]:
```
sorted(t8)
```

Out[44]: [1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 5, 5, 6, 7, 8, 9, 10]

In [45]:
```
enumerate(t8)# adds counter(similar to index)
```

Out[45]: <enumerate at 0x1d1234ea480>

In [46]:
```
for i in enumerate(t8):
    print(i)
```

```
(0, 1)
(1, 2)
(2, 3)
(3, 4)
(4, 2)
(5, 3)
(6, 1)
(7, 4)
(8, 5)
(9, 2)
(10, 3)
(11, 5)
(12, 6)
(13, 7)
(14, 8)
(15, 9)
(16, 10)
```

Accessing Elements

Slicing

In [48]:
```
t8[:]
```

Out[48]: (1, 2, 3, 4, 2, 3, 1, 4, 5, 2, 3, 5, 6, 7, 8, 9, 10)

In [49]:
```
t8[::]
```

Out[49]: (1, 2, 3, 4, 2, 3, 1, 4, 5, 2, 3, 5, 6, 7, 8, 9, 10)

In [50]:
```
t8[0:]
```

Out[50]: (1, 2, 3, 4, 2, 3, 1, 4, 5, 2, 3, 5, 6, 7, 8, 9, 10)

In [51]:
```
t8[:-1]
```

Out[51]: (1, 2, 3, 4, 2, 3, 1, 4, 5, 2, 3, 5, 6, 7, 8, 9)

In [52]:
```
t8[::-1]
```

Out[52]: (10, 9, 8, 7, 6, 5, 3, 2, 5, 4, 1, 3, 2, 4, 3, 2, 1)

In [53]:
```
t8[::-2]
```

Out[53]: (10, 8, 6, 3, 5, 1, 2, 3, 1)

In [54]:
```
t8[::-3]
```

Out[54]: (10, 7, 3, 4, 2, 2)

In [55]:
```
t8[0:10:2]
```

Out[55]: (1, 3, 2, 1, 5)

In [56]:
```
t8[10:0:2]
```

Out[56]: ()

In [57]:
```
t8[2:8:-1]
```

Out[57]: ()

In [62]:
```
t8[8:2:-1]
```

Out[62]: (5, 4, 1, 3, 2, 4)

In [63]:
```
t8[0:-1]
```

Out[63]: (1, 2, 3, 4, 2, 3, 1, 4, 5, 2, 3, 5, 6, 7, 8, 9)

In [64]:
```
t8[0:-4]
```

Out[64]: (1, 2, 3, 4, 2, 3, 1, 4, 5, 2, 3, 5, 6)

In [67]:
```
t8[0:]
```

Out[67]: (1, 2, 3, 4, 2, 3, 1, 4, 5, 2, 3, 5, 6, 7, 8, 9, 10)

In [68]:
```
t8[2:]
```

Out[68]:
```
(3, 4, 2, 3, 1, 4, 5, 2, 3, 5, 6, 7, 8, 9, 10)
```

In [69]:
```
t8[-2:]
```

Out[69]:
```
(9, 10)
```

INDEXING

In [70]:
```
t8[10]
```

Out[70]:
```
3
```

In [71]:
```
t8[2]
```

Out[71]:
```
3
```

In [72]:
```
t8[-8]
```

Out[72]:
```
2
```

In [73]:
```
t8[-3]
```

Out[73]:
```
8
```

In [74]:
```
t8[0]
```

Out[74]:
```
1
```

In [75]:
```
t8[1]
```

Out[75]:
```
2
```

In [76]:
```
t8[-1]
```

Out[76]:
```
10
```

Concatenation & Repeatation

In [81]:
```
# Concatenation is done using (+) operator
# it is used to add one tuple to end of another tuple

tup1 = (1,2,3,4,5)
tup2 = (6,7,8,9,10)
tup1 + tup2
```

Out[81]:
```
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

In [ ]:
```
# unlike append it unpacks the items from object adds at end of other list
# just similar to extend()
```

In [83]:
```
tup3 = (1,2,3)
tup3 + tup2
```

Out[83]:
```
(1, 2, 3, 6, 7, 8, 9, 10)
```

In [84]:
```
tup4 = tup1+tup3+tup2
```

In [85]:
```
tup4
```

Out[85]:
```
(1, 2, 3, 4, 5, 1, 2, 3, 6, 7, 8, 9, 10)
```

In [86]:
```
# Repeatation is done using (*) operator
# it is used to repeat the tuple no.of times

tup3 *2
```

Out[86]:
```
(1, 2, 3, 1, 2, 3)
```

In [87]:
```
tup3 * 5
```

Out[87]:
```
(1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3)
```

In [88]:
```
tup5 = tup4 * 2
```

In [89]:
```
tup5
```

Out[89]:
```
(1,
 2,
 3,
 4,
 5,
 1,
 2,
 3,
 6,
 7,
 8,
 9,
 10,
 1,
 2,
 3,
 4,
 5,
 1,
 2,
 3,
 6,
 7,
 8,
 9,
 10)
```

LOOP

In [91]:
```
for i in tup1:
    print(i)
```

```
1
2
3
4
5
```

In [92]:
```
for i in enumerate(tup1):
    print(i)
```

```
(0, 1)
(1, 2)
(2, 3)
(3, 4)
(4, 5)
```

In [93]:
```
for i in tup1:
    print(i*2)
```

```
2
4
6
8
10
```

In [94]:
```
for i in tup1:
    print(i+2)
```

```
3
4
5
6
7
```

In [95]:
```
for i in tup1:
    print(i/2)
```

```
0.5
1.0
1.5
2.0
2.5
```

In [96]:
```
for i in tup1:
    print(i//2)
```

```
0
1
1
2
2
```

MEMBERSHIP OPERATOR

In [ ]:
```
# Membership operators
#>>in
#>>not in
# as the name they check membership of values in datastructures
```

```
#question>>a in b
# returns True if value in ds
# returns flase if not in ds

#question>>a not in b
# returns True if value not in ds
# returns flase if in ds
```

In [97]:
```
a = (100,200,300,500)
400 in a
```

Out[97]: False

200 in a

In [102…
```
400 not in a
```

Out[102… True

In [103…
```
200 not in a
```

Out[103… False

In [104…
```
# we can also work with conditional statments

if 100 in a:
    print("hi")
else:
    print("hello")
```

hi

In [105…
```
if 100 not in a:
    print("hi")
else:
    print("hello")
```

hello

CONVERTING TUPLE TO OTHER TYPES(IF ANY CHANGES)

In [110…
```
tup = [1,2,3,4]
type(tup)
```

Out[110… list

In [111…
```
tuple(tup)
```

Out[111… (1, 2, 3, 4)

In [113…
```
tup5
```

Out[113…
```
(1,
 2,
 3,
 4,
 5,
 1,
 2,
 3,
 6,
 7,
 8,
 9,
 10,
 1,
 2,
 3,
 4,
 5,
 1,
 2,
 3,
 6,
 7,
 8,
 9,
 10)
```

In [116…
```
tp = list(tup5)
tp
```

Out[116…
```
[1,
 2,
 3,
 4,
 5,
 1,
 2,
 3,
 6,
 7,
 8,
 9,
 10,
 1,
 2,
 3,
 4,
 5,
 1,
 2,
 3,
 6,
 7,
 8,
 9,
 10]
```

In [120…
```
tp.remove(1)
```

In [121…
```
tp.pop()
```
Out[121… 10

In [122…
```
tp.pop()
```
Out[122… 9

In [123…
```
tp.pop()
```
Out[123… 8

In [124…
```
tp.pop()
```
Out[124… 7

In [125…
```
tp.pop()
```
Out[125… 6

In [126…
```
tp.pop()
```
Out[126… 3

In [127…
```
tp.pop()
```
Out[127… 2

In [128…
```
tp.pop()
```
Out[128… 1

In [129…
```
tp
```
Out[129… [2, 3, 4, 5, 1, 2, 3, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5]

In [130…
```
tp.insert(0,1)
```

In [131…
```
tp
```
Out[131… [1, 2, 3, 4, 5, 1, 2, 3, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5]

In [132…
```
tp.pop()
```
Out[132… 5

In [133…
```
tp.pop()
```
Out[133… 4

In [134…
```
tp.pop()
```
Out[134… 3

In [135…
```
tp.pop()
```

Out[135… 2

In [136…
```
tp.pop()
```

Out[136… 1

In [137…
```
tp
```

Out[137… [1, 2, 3, 4, 5, 1, 2, 3, 6, 7, 8, 9, 10]

In [138…
```
n_t = tuple(tp)
```

In [139…
```
n_t
```

Out[139… (1, 2, 3, 4, 5, 1, 2, 3, 6, 7, 8, 9, 10)

In [142…
```
str = 'python'
list = [1,2,3,4]
set = {'hi','hello'}
dict = {'k':'v'}
print(tuple(str))
print(tuple(list))
print(tuple(set))
print(tuple(dict.items()))
```

```
('p', 'y', 't', 'h', 'o', 'n')
(1, 2, 3, 4)
('hello', 'hi')
(('k', 'v'),)
```

In [ ]:
```
# we can see as we gave only one value to dict
# it added TRAILERING comma to tuple(dict.items())

# one more thing it converted every character of string individually to tuple
```

Unpacking

In [146…
```
# you can extract values back in to variables

lang = ('tel','eng','hin')
(cse,csm,css) = lang
print(cse)
print(csm)
print(css)
```

```
tel
eng
hin
```

In [148…
```
print(lang)
```

```
('tel', 'eng', 'hin')
```

In [153…
```
lang = ('tel','eng','hin')
(cse,csm,css) = lang
print(cse)
```

```
print(csm)
print(css)
```

```
tel
eng
hin
```

In [ ]: