Module 10

Tuples ()

So, indexing is possible.

- Like List, tuples are also (Ordered) collection of data
- Unchangeable ← Lists are changeable whereas Tuples are unchangeable.
- Iterable
- Can be heterogenous.

```
planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter",
           "Saturn", "Uranus", "Neptune", "Pluto"]
type(planets)
list
planets[2] = "Uvaraj"
planets
['Mercury', 'Venus', 'Uvaraj', 'Mars', 'Jupiter', 'Saturn',
 'Uranus', 'Neptune', 'pluto']
```

?

```
t = ("Mercury", "Venus", "Earth", "Mars", "Jupiter",
     "Saturn", "Uranus", "Neptune", "Pluto")
type(t)
tuple
# indexing
t[0]
'Mercury'
t[-1]
'Pluto'
# slicing
t[1:4]
('Venus', 'Earth', 'Mars')
```

Unchangeable (immutable)

```
t[2] = "Uvaraj"
TypeError : 'tuple' object does not support item assignment
```

Heterogenous

```
t = ("Uvaraj", 2, 2414, True)
type(t)
tuple
type(t[0])
str
type(t[1])
int
```

Creating a Tuple

```
t = ()
type(t)
tuple

t1 = tuple()
type(t1)
tuple

print(t, t1)
() ()
```

① ② Two ways of creating empty tuple.

```
t3 = (2)
type(t3)
int

t4 = (2,)
type(t4)
tuple
t4
(2,)
```

⚠ Caution while creating a Tuple with only one element.

```
t5 = tuple("Rahul")   <-- iterable
t5
('R', 'a', 'h', 'u', 'l')
type(t5)
tuple
```

```
t6 = (23, 5, 9, 12)
len(t6)
4
```

```
t7 = ([1,2], [4,5])   <-- Tuple with lists inside of it.
print(len(t7))
2
print(type(t7))
<class 'tuple'>
print(t7)
([1,2], [4,5])
t7[0]
[1,2]
t7[1]
[4,5]
type(t7[0])
list
```

Mutability in Tuples

```
t = (2,3,4)
t[0] = 45   <-- Tuples are immutable
TypeError: 'tuple' object does not support item assignment
```

```
t1 = ([4,3,4,5], "Uvaraj")
type(t1)
tuple
type(t1[0])
list
type(t1[1])
str
```

Individual elements.

```
t1[0]
[1, 3, 4, 5]
t1[0][0]
1
t1[0][0] = 45
t1
([45, 3, 4, 5], 'Uvaraj')
```

Tuple unpacking

```
a = 3
b = 4
c = 5
d = 1
print(a, b, c, d)
3 4 5 1
```

```
a, b = 2, 4
print(a, b)
2, 4
```

```
t = (1, 2, 3, 4)
a, b, c, d = t
print(a, b, c, d)
1, 2, 3, 4
```

This process is known as Unpacking. This is useful to return multiple values (one tuple) from a function.

Tuple operations

```
t = (1, 2, 3, 4)
t.count ?
```

① COUNT

```
Signature: t.count(value, /)
Docstring: Return number of occurences of value.
Type:      Builtin_function_or_method
t.count(1)
1
t.count(45)
0
```

---

t.index ?

② INDEX

Signature : t.index (value, start = 0, stop = 9223372..., /)

Docstring : Return first index of value.
Raises ValueError if the value is not present.

Type : builtin_function_or_method.

```
t.index(2)
1
```

---

③ ITERATION

```
t
(1, 2, 3, 4)
len(t)
4
for i in t:
    print(i, end = " ")
1 2 3 4
for i in t:
    print(i**2, end = " ")
1 4 9 16
```

---

④ CONCATENATION using + and *

```
t
(1, 2, 3, 4)
t1 = (5, 6)
t2 = t + t1
t2
(1, 2, 3, 4, 5, 6)
t2 * 2
(1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6)
t2
(1, 2, 3, 4, 5, 6)
```

---

⑤ TUPLE ⟷ LIST CONVERSION

```
t
(1, 2, 3, 4)
lst = list(t)        ← Tuple to list
type(lst)
list
```

```
tup = tuple(lst)
type(tup)
tuple
```

Challenges

① Get the index

Write a function to search for a given element "elem" in the tuple "tup", and return its index. Return -1 if elem is not present in tup.

Input format:

The first line consists of tuple.
The second line consists of an element for which index is required to be returned.

Output format:

Return the index value in integer format.

Sample input:

(10, 20, 30, 40, 50)
30

Sample output:

2

Explanation:

30 is present at index 2, therefore 2 is returned.

```
def get_index(tup, elem):
    if elem in tup:
        return tup.index(elem)
    else:
        return -1

tup = (10, 20, 30, 40, 50)
elem = 30
```

2

② Divide the Tuple.

Write a program to divide a given tuple into two tuples that contain even and odd indexed elements of the original tuple. Print both these tuples in the given output format.

Note: The input tuple follows 1-based indexing. This means the element at index 0 is treated as having index as 1.

For example: (9, 2, 3). The 1-based index of 2 is 2.

Input Format:

The input contains a tuple as an argument to the function.

Output Format:

Print two tuples, one for odd indexed elements and another for even indexed elements in the following format.

odd: (.....)
Even: (.....)

Sample input

(10, 8, 5, 2, 10, 15, 10, 8, 5, 8, 8, 2)

Sample output

odd: (10, 5, 10, 10, 5, 8)
Even: (8, 2, 15, 8, 8, 2)

```
def odd_even_split_tuple(tup):
    # Odd-indexed elements (1-based indexing)
    odd = tup[0::2]

    # Even-indexed elements (1-based indexing)
    even = tup[1::2]

    # Print the result in the required format
    print(f"odd: {odd}")
    print(f"Even: {even}")
```

③ What is the type of the following "fruits" variable ?

```
fruits = ("Orange")
```

str

④ Predict the output of the following code, as we are trying to append to an immutable tuple :

```
name_lst = ["Vijay", "Vicky"]
tup = ("Item_1", 0.5, name_lst)
name_lst.append ("Vishal")
print (tup)
```

("Item_1", 0.5, ["Vijay", "Vicky", "Vishal"])

⑤ What is the output of the following ?

```
elements = (10, 20, 30, 40, 50, 60, 70, 80)
print (elements [2:5], elements [:4], elements [3:100])
```

(30, 40, 50) (10, 20, 30, 40) (40, 50, 60, 70, 80)