Tuples in Python

Problem

A hospital has received a set of lab reports. Totally five tests are conducted in the lab and the report is prepared. The report consist of name of the test and the value observed for that particular patient. Given the details of a test made for a patient, write an algorithm and the subsequent Python program to print if the test result is normal or not normal by referring the values in Table 1. Since the value is sensitive, provide a mechanism so that the values do not get altered.

Name of the Test	Minimum Value	Maximum Value
Test1	20	30
Test2	35.5	40
Test3	12	15
Test4	120	150
Test5	80	120

PAC For Lab Test Problem

Input	Processing	Output
Test name and a pair of values indicating the minimum and the maximum value for the five tests Name of the test and the value observed	Check if the observed value is in the range of permissible values for the test and print 'Normal' or 'Abnormal'	Print 'Normal' or 'Abnormal'

Pseudocode LabTest Problem

```
FOR i =0 to 5
        READ test_Name_i
        READ minimum_i
        READ maximum_i
        Map test_Name_i to minimum_i and maximum_i
READ test_Name_Chk
READ observed_Value
END_FOR
```

```
IF observed_Value>min of test_Name_Chk
    and observed_Value <max of test_Name_Chk
            THEN
                    PRINT 'Normal'
                ELSE
                    PRINT 'Abnormal'
END IF
```

Store the values such that it is not getting modified

We Already Know

- To read values
- Map a value to another - Dictionary
- Print Values
- Form a pair of values – List – But the values can be changed

- Yet to learn about pairing values that cannot be modified

Tuples

- A tuple is a collection which is **ordered and unchangeable.**
- Sequence of **immutable** Python objects

- Tuples cannot be changed unlike lists and tuples use **parentheses**, whereas lists use square brackets.

- Creating a tuple is as simple as putting different comma-separated values.
- Optionally you can put these **comma-separated values** between parentheses also.
- For example –

```
tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5 )
tup3 = "a", "b", "c", "d"
```

empty tuple

tup1 = ()

To write a tuple containing a single value you have to include a comma –

a = (50) # an integer

tup1 = (50,) # tuple containing an integer

- **tuple indices start at 0**

print ("tup1[0]: ", tup1[0]) # print physics

print ("tup2[1:5]: ", tup2[1:5]) # print (2,3,4,5)

Tuples in Action

```
>>> (1, 2) + (3, 4)  # Concatenation
(1, 2, 3, 4)

>>> (1, 2) * 4       # Repetition
(1, 2, 1, 2, 1, 2, 1, 2)

>>> T = (1, 2, 3, 4)  # Indexing, slicing

>>> T[0], T[1:3]
(1, (2, 3))
```

Sorted method in Tuples

```
>>> tmp = ['aa', 'bb', 'cc', 'dd']

>>> T = tuple(tmp)

 # Make a tuple from the list's items

>>> T
('aa', 'bb', 'cc', 'dd')

>>> sorted(T)   #Return a list of items
# Since tuples are immutable

['aa', 'bb', 'cc', 'dd']
```

List comprehensions can also be used with tuples.

The following, for example, makes a list from a tuple, adding 20 to each item along the way:

```
>>> T = (1, 2, 3, 4, 5)

>>> L = [x + 20 for x in T]
```

Equivalent to:

```
>>>L = []
>>>for x in T:
        L.append(x+20)
>>> L
[21, 22, 23, 24, 25]
```

Index method can be used to find the position of particular value in the tuple.

```
>>> T = (1, 2, 3, 2, 4, 2)
>>> T.index(2)          # Offset of first appearance of 2
1
>>> T.index(2, 2)       # Offset of appearance after offset 2
3
>>> T.count(2)          # How many 2s are there?
3
```

Nested Tuples

```
>>> T = (1, [2, 3], 4)

>>> T[1] = 'spam'
# fails: can't change tuple itself TypeError: object doesn't support item
assignment

>>> T[1][0] = 'spam'
# Works: can change mutables inside immutable object

>>> T
(1, ['spam', 3], 4)
```

```
>>> bob = ('Bob', 40.5, ['dev', 'mgr'])
# Tuple record

>>> bob
('Bob', 40.5, ['dev', 'mgr'])

>>> bob[0], bob[2]        # Access by position
('Bob', ['dev', 'mgr'])
```

Prepares a Dictionary record from tuple

```
>>> bob = dict(name='Bob', age=40.5, jobs=['dev', 'mgr'])

>>> bob
{'jobs': ['dev', 'mgr'], 'name': 'Bob', 'age': 40.5}

>>> bob['name'], bob['jobs']          # Access by key
('Bob', ['dev', 'mgr'])
```

Dictionary to Tuple

We can convert **parts of dictionary to a tuple** if needed:

```
>>> tuple(bob.values())                    # Values to tuple
    (['dev', 'mgr'], 'Bob', 40.5)

>>> list(bob.items())                      # Items to tuple
```

list [('jobs', ['dev', 'mgr']), ('name', 'Bob'), ('age', 40.5)]

Using Tuples

- Immutable which means you **cannot update or change** the values of tuple elements:

```
tup1 = (12, 34.56)
tup2 = ('abc', 'xyz')

# Following action is not valid for tuples
tup1[0] = 100
```

- You are **able to take portions of existing tuples to create new tuples** as the following example demonstrates

```
tup3 = tup1 + tup2
print(tup3)
```

Delete Tuple Elements

- **Removing individual** tuple elements is **not possible**
- But possible to **remove** an **entire tuple**.

```
tup = ('physics', 'chemistry', 1997, 2000)

print (tup)
del tup;
print ("After deleting tup : ")
print (tup)
```

Packing and unpacking

```
t = ('foo', 'bar', 'baz', 'qux')
>>> t
('foo', 'bar', 'baz', 'qux')
>>> t[0]
'foo'
>>> t[-1]
'qux'
```

- If that "packed" object is subsequently assigned to a new tuple, the individual items are "unpacked" into the objects in the tuple:

```
>>> (s1, s2, s3, s4) = t
>>> s1
'foo'
>>> s2
'bar'
>>> s3
'baz'
```

Packing and unpacking

```
>>>(s1, s2, s3) = t
Traceback (most recent call last):
File "<pyshell#16>", line 1, in <module>
 (s1, s2, s3) = t
ValueError: too many values to unpack (expected 3)

>>> (s1, s2, s3, s4, s5) = t
Traceback (most recent call last):
 File "<pyshell#17>", line 1, in <module>
 (s1, s2, s3, s4, s5) = t
ValueError: not enough values to unpack (expected 5, got 4)
```

Packing and unpacking

- Packing and unpacking can be combined into one statement to make a compound assignment:

```
>>> (s1, s2, s3, s4) = ('foo', 'bar', 'baz', 'qux')
 >>> s1
 'foo'
>>> s2
'bar'
>>> s3
'baz'
>>> s4
 'qux'
```

- The number of elements in the tuple on the left of the assignment must equal the number on the right.

Basic Tuples Operations

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Indexing, Slicing

If L = ('spam', 'Spam', 'SPAM!')

Python Expression	Results	Description
L[2]	'SPAM!'	Offsets start at zero
L[-2]	'Spam'	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

Built-in Tuple Functions

```
tuple1, tuple2 = (123, 'xyz'), (456, 'abc')
```

len(tuple1)

2

When we have numerical tuple:

```
t1 = (1,2,3,7,4)
```

max(t1)

min(t1)

Converts a list into a tuple

tuple(seq)

```
t2=tuple([2,4])

>>> t2
(2, 4)
```

Problem:

An University has published the results of the term end examination conducted in April. List of failures in physics, mathematics, chemistry and computer science is available. Write a program to find the number of failures in the examination

Pseudocode

```
READ maths_failure, physics_failure, chemistry_failure and cs_failure
SET failure to empty
FOR each item in maths_failure
        ADD item to failure
FOR each item in physics_failure
        IF item is not in failure THEN
                ADD item to failure
        END IF
FOR each item in chemistry_failure
        IF item is not in failure THEN
                ADD item to failure
        END IF
FOR each item in cs_failure
        IF item is not in failure THEN
                ADD item to failure
        END IF
SET count = 0
FOR each item in failure
        count = count + 1
PRINT count
```

```
lab_Reading = {}
for i in range(0,5):
    test_Name = input()
    min = float(input())
    max = float(input())
    lab_Reading[test_Name] = (min,max)
print(lab_Reading)
#Read name of test
chk_Test = input()
#read observed value of test
obs_Value = float(input())
#find range of values for the specified test
range_Test = lab_Reading[chk_Test]
min = range_Test[0]
max = range_Test[1]
if min<obs_Value<max:
    print('Normal')
else:
    print ('Abnormal')
```