VIT UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
VELLORE ▪ CHENNAI
www.vit.ac.in

Problem Solving and Programming CSE1001

Prof. Tulasi Prasad Sariki

September 17, 2019

Tuples and Sets

Problem

A hospital has received a set of lab reports. Totally five tests are conducted in the lab and the report is prepared in such a way that the n^{th} number correspond to value of $test_n$. 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. 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 - For Lab Test 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

Already we know

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

Already we 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

- Sequence of immutable Python objects
- Tuples cannot be changed like 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";

Working with Tuples

- 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))

Sorting 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) → ('aa', 'bb', 'cc', 'dd')

List comprehensions can also be used with tuples.

- 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
- >>> 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 list of tuples
  [('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)
  Error

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!'	Offset start at zero
L[-2]	'Spam'	Negative count from right
L[1:]	['Spam','SPAM!']	Slicing Fetches Section

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) # prints 7
- >>> min(t1) # prints 1

Converts a list into a tuple

- tuple(seq)
- >>> t2=tuple([2,4])
- >>> t2
  (2, 4)

Python code for Lab Test Problem

```
lab_Reading={}
for i in range(0,5):
    test_Name=input('Enter test Name')
    min=float(input('Enter min value'))
    max=float(input('Enter max value'))
    lab_Reading[test_Name]=(min,max)
print(tuple(lab_Reading.items()))
chk_Test=input('Enter check test') #Read Name of Test
#Read Observed value of Test
obs_Value=float(input('Enter obseved value'))
#Find range of speciefied 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')
```

Problem: University Result

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. This includes the count of failures in one or more subjects

Problem: University Result

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. This includes the count of failures in one or more subjects

PAC For University Result Problem

Input	Processing	Output
Read the register number of failures in Maths, Physics, Chemistry and Computer Science	Create a list of register numbers who have failed in one or more subjects Count the count of failures	Print Count

Pseudocode

```
READ maths_failure, physics_failure, chemistry_failure and cs_failure
Let failure be 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
```

Sets

- an unordered collection of unique and immutable value element that supports operations corresponding to mathematical set theory
- Set is mutable
- No duplicates
- Sets are iterable, can grow and shrink on demand, and may contain a variety of object types
- Does not support indexing
- >>> x = {1, 2, 3, 4}
- >>> y = {'apple','ball','cat'}
- >>> x1 = set('spam') # Prepare set from a string
- >>> print (x1)
  {'s', 'a', 'p', 'm'}
- >>>x1.add('alot') # Add an element to the set
- >>> print (x1)
  {'s', 'a', 'p', 'alot', 'm'}

Set Operations

- >>> S1 = {1, 2, 3, 4}
- Union (|)
- >>> S2 = {1, 5, 3, 6} | S1
- >>> print(S2) # prints {1, 2, 3, 4, 5, 6}
- Intersection (&)
- >>> S2 = S1 & {1, 3}
- >>> print(S2) # prints {1, 3}
- Difference (-)
- >>> S2 = S1 - {1, 3, 4}
- >>> print(S2) # prints {2}
- Super set (>)
- >>> S2 = S1 > {1, 3}
- >>> print(S2) # prints True
- Empty sets must be created with the set built-in, and print the same way

Set Operations

- >>> S2 = S1 - {1, 2, 3, 4}
- >>> print(S2) # prints set() - Empty set
- Empty curly braces represent empty dictionary but not set
- In interactive mode - type({}) gives
  <class 'dict'>
- >>> {1, 2, 3} | {3, 4}
  {1, 2, 3, 4}
- >>> {1, 2, 3} | [3, 4]
  TypeError: unsupported operand type(s) for |: 'set' and 'list'
- >>> {1, 2, 3} | set([3, 4]) #Convert list to set and work
  {1,2,3,4}
- >>> {1, 2, 3}.union([3, 4])
  {1,2,3,4}
- >>> {1, 2, 3}.union({3, 4})
  {1,2,3,4}

Immutable constraints and frozen sets

- Can only contain immutable (a.k.a. "hashable") object types
- lists and dictionaries cannot be embedded in sets, but tuples can if you need to store compound values.
- Tuples compare by their full values when used in set operations:
- >>> S = {1.23}
- >>> S.add([1, 2, 3])
  TypeError: unhashable type: 'list'
- >>> S.add({'a':1})
  TypeError: unhashable type: 'dict'
- Works for tuples:
- >>> S.add((1, 2, 3))
- >>> S
  {1.23, (1, 2, 3)}

- >>> S | {(4, 5, 6), (1, 2, 3)}
  {1.23, (4, 5, 6), (1, 2, 3)}
- >>> (1, 2, 3) in S # Check for tuple as a whole
  True
- >>> (1, 4, 3) in S
  False

clear()

- All elements will removed from a set.
- >>> cities = {"Stuttgart", "Konstanz", "Freiburg"}
- >>> cities.clear()
- >>> cities
- set() # empty
- >>>

Copy

- Creates a shallow copy, which is returned.
- >>> more_cities = {"Winterthur","Schaffhausen","St. Gallen"}
- >>> cities_backup = more_cities.copy()
- >>> more_cities.clear()
- >>> cities_backup # copied value is still available
  {'St. Gallen', 'Winterthur', 'Schaffhausen'}
- Just in case, you might think, an assignment might be enough:
- >>> more_cities = {"Winterthur","Schaffhausen","St. Gallen"}
- >>> cities_backup = more_cities #creates alias name
- >>> more_cities.clear()
- >>> cities_backup
  set()
- >>>
- The assignment "cities_backup = more_cities" just creates a pointer, i.e. another name, to the same data structure.

difference_update()

- removes all elements of another set from this set. x.difference_update() is the same as "x = x - y"
- >>> x = {"a","b","c","d","e"}
- >>> y = {"b","c"}
- >>> x.difference_update(y)
- >>> x → {'a', 'e', 'd'}

discard(el)

- el will be removed from the set, if it is contained in the set and nothing will be done otherwise
- >>> x = {"a","b","c","d","e"}
- >>> x.discard("a")
- >>> x → {'c', 'b', 'e', 'd'}
- >>> x.discard("z")
- >>> x → {'c', 'b', 'e', 'd'}

remove(el)

- works like discard(), but if el is not a member of the set, a KeyError will be raised.
- >>> x = {"a","b","c","d","e"}
- >>> x.remove("a")
- >>> x
  {'c', 'b', 'e', 'd'}
- >>> x.remove("z")
  Traceback (most recent call last): File "<stdin>", line 1, in <module> KeyError: 'z

isdisjoint()

- This method returns True if two sets have a null intersection

issubset()

- x.issubset(y) returns True, if x is a subset of y
- "<=" is an abbreviation for "Subset of" and ">=" for "superset of"
- "<" is used to check if a set is a proper subset of a set

issuperset()

- x.issuperset(y) returns True, if x is a superset of y. ">=" - abbreviation for "issuperset of
- ">" - to check if a set is a proper superset of a set
- >>> x = {"a","b","c","d","e"}
- >>> y = {"c","d"}
- >>> x.issuperset(y) → True
- >>> x > y → True
- >>> x >= y → True
- >>> x >= x → True
- >>> x > x → False
- >>> x.issuperset(x) → True
- >>> x = {"a","b","c","d","e"}
- >>> y = {"c","d"}
- >>> x.issubset(y)→ False
- >>> y.issubset(x) → True

- >>> x = {"a","b","c","d","e"}
- >>> y = {"c","d"}
- >>> x.issubset(y)→ False
- >>> y.issubset(x) → True
- >>> x < y → False
- >>> y < x # y is a proper subset of x
  True
- >>> x < x # a set is not a proper subset of oneself.
  False
- >>> x <= x → True

pop()

- pop() removes and returns an arbitrary set element.
- The method raises a KeyError if the set is empty
- >>> x = {"a","b","c","d","e"}
- >>> x.pop()
  'a'
- >>> x.pop()
  'c'

frozenset

- Sets themselves are mutable too, and so cannot be nested in other sets directly;
- if you need to store a set inside another set, the frozenset built-in call works just like set but creates an immutable set that cannot change and thus can be embedded in other sets

To create frozenset:

- >>> cities = frozenset(["Frankfurt", "Basel","Freiburg"])
- >>> cities.add("Strasbourg") # cannot modify
  Traceback (most recent call last): File "<stdin>", line 1, in <module> AttributeError: 'frozenset' object has no attribute 'add'

Set comprehensions

- run a loop and collect the result of an expression on each iteration
- result is a new set you create by running the code, with all the normal set behavior
- >>> {x ** 2 for x in [1, 2, 3, 4]}
  {16, 1, 4, 9}
- >>> {x for x in 'spam'}
  {'m', 's', 'p', 'a'}
- >>> S = {c * 4 for c in 'spam'}
- >>> print(S)
  {'pppp','aaaa','ssss', 'mmmm'}
- >>> S = {c * 4 for c in 'spamham'}
  {'pppp','aaaa','ssss', 'mmmm','hhhh'}
- >>> S | {'mmmm', 'xxxx'}
  {'pppp', 'xxxx', 'mmmm', 'aaaa', 'ssss'}
- >>> S & {'mmmm', 'xxxx'}
  {'mmmm'}

Python code for University Result

```
math=set()
phy=set()
che=set()
cs=set()
m_N=int(input("Enter Math No"))
for i in range(0, m_N):
    val=input("Enter Regno")
    math=math | {val}
m_P=int(input("Enter Physics No"))
for i in range(0, m_P):
    val=input("Enter Regno")
    phy=phy | {val}
```

Python code for University Result - Contd...

```
m_C=int(input("Enter Chemistry No"))
for i in range(0, m_C):
    val=input("Enter Regno")
    che=che | {val}
m_CS=int(input("Enter Computer No"))
for i in range(0, m_CS):
    val=input("Enter Regno")
    cs=cs | {val}
failure = math|phy|che|cs
print(len(failure))
```

Exercise 1

While John got ready for the office a shopping list was given by his son and daughter. While returning home John decides to buy the items from a shop. He decides to buy the items common in both the list, then the items listed only by his son and at last the items listed only by his daughter. Write a program to help him in achieving the task.

Exercise 2

A marketing company has branch in a set of cities S. The company sends three of their sales man to various cities in set S. In the middle of the year, help the manager to find out the cities that are already visited by the sales men and the cities that are yet to be visited.

Exercise 3

A proctor of our university wants a student to write a proram that calculates the average of marks scored by her wards in CAT1. She has the details such as name, register number, mark1, mark2 and mark3 of her proctees. The constraint given by the faculty is that any of the details must not be altered by mistake. Help the student to develop a Python program

Exercise 4

A super market plan to conduct a surprise cum lucky winner contest in a different way. They decide to give a prize for two customers with maximum number of items in common in their shopping list. Write a program to identify common items from the shopping list of two customers. Prepare the common items list as read only.

Exercise 5

Write a program to maintain a telephone directory of the employees of an organization. If the employee has more than one number store all the numbers. Write a program to print the mobile numbers given full or part of the name of the employee. Eg: Given name of the employee as John the program must print phone numbers of John Paul and Michel John.

Exercise 6

Write a program to store the name of the players against each of a 20-20 cricket team. The program should print the name of the players given the team name.


Thank you