**Lab 5: List and Tuple objects**

Learning Outcomes:

* Get familiar with using list and tuple sequences
* Get to know to modify list object, add, remove elements from list
* Get to solve problems using lists and tuples

Instructions:

* Suggest that you create a working folder **is111\lab5** in your **C** or **D** drive. Store all your solutions in this working folder.
* Challenging questions are marked with \*.

To submit:

* Please submit your working solutions via your assignment Dropbox in eLearn **within 1 week**. The Dropbox will be closed after the due date.
* Zip up all your source files into a single zip file called **<your email ID>\_lab5.zip** (e.g. **ahlian.lim.2011\_lab5.zip**). You should only submit a single zip file for each lab.

1. Examine **lab5\_1.py** given to you below. Complete the function get\_numbers to return a new list that contains only integers between min and max parameters. Your code should not modify the original list.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15 | def get\_numbers(num\_list, min, max):  list1 = [4,10,12,28,24,18,5,20,15,21,30,22,21,14,17,28,26,24,6,8,15]  new\_list = get\_numbers(list1, 10, 20)  print("Original list", list1)  print("After function is called", new\_list) |

Here is the output when **lab5\_1** is run (with the given test code):

|  |
| --- |
| D:\is111\lab5>**python lab5\_1.py**  Original list [4, 10, 12, 28, 24, 18, 5, 20, 15, 21, 30, 22, 21, 14, 17, 28, 26, 24, 6, 8, 15]  After function is called [10, 12, 18, 20, 15, 14, 17, 15] |

To submit**: lab5\_1.py**

1. \*Write a program in a file named **lab5\_2.py** that gets 10 integer inputs from the user. The program should then display the minimum, maximum and median of all numbers entered.

Note: The median is the middle of list of numbers. For example, median of numbers 12, 4, 5 is 5. In case of odd amount of numbers, the median is the exact middle number of numbers when arranged sorted.

In case of even amount of numbers, we would get a pair of middle numbers. The median is half way between them. As an example, median of numbers 6, 12, 4, 10 is 8 (6 + 10) / 2 because when placed in order the middle numbers would be 6 and 10.

Hint:  
1. This is how you can sort a list.

>>> numbers = [4, 10, 1, 3, -2]

>>> numbers.sort()

>>> numbers

[-2, 1, 3, 4, 10]

Here are some sample outputs when **lab5\_2** is run:

|  |
| --- |
| D:\is111\lab5>**python lab5\_2.py**  Enter num 1 :-9  Enter num 2 :4  Enter num 3 :-11  Enter num 4 :19  Enter num 5 :5  Enter num 6 :17  Enter num 7 :4  Enter num 8 :2  Enter num 9 :8  Enter num 10 :1  Minimum of numbers: -11  Maximum of numbers: 19  Median of numbers: 4.0  D:\is111\lab5> |

To submit**: lab5\_2.py**

1. This exercise requires you to trace through the code. Print the output of the following two programs.

(a)

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | def fun1(str\_list):  str\_list.insert(1,"durian")  out = []  for i in str\_list:  if len(i) > 3:  out.append(i)  return out    list1 = ["one", "apple", "six", "oranges", "bunch", "grapes"]  print ("Before :", list1)  #calling the function fun1  new\_list = fun1(list1)  print("After :", list1)  print("New List:", new\_list) |

(b)

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14 | def fun2(list\_2d):  list\_2d.insert (1, [7,8])  new\_list = list\_2d[1:]  new\_list[1] = [10,11]  return new\_list    list1 = [[1,2], [3,4]]  list2 = [5,6]  list1.append(list2)  print ("Before :", list1)  list2 = fun2(list1)  print ("After :", list1)  print ("List2 :", list2) |

To submit**: -**

1. \* In a file named **lab5\_4.py** write a function called count\_numbers that takes in a list containing numbers and returns the count of numbers in the list. Your program has to cater to the possibility of having a list of numbers as element(s) in the input parameter. (How to deal with the possibility of numerous list?)

Here is the output when **lab5\_4** is run calling the function as:

print("Count of numbers", count\_numbers([4,6,8,10,-3]))

|  |
| --- |
| D:\is111\lab5>**python lab5\_4.py**  Count of numbers:5 |

Here is the output when **lab5\_4** is run calling the function as:

print("Count of numbers", count\_numbers([4,6,[1,2],10,[-1,-3]]))

|  |
| --- |
| D:\is111\lab5>**python lab5\_4.py**  Count of numbers:7 |

To submit**: lab5\_4.py**

1. In a file named **lab5\_5.py** write a function called merge\_lists that takes in 2 lists containing numbers and returns a new list containing all unique numbers present in the two lists sorted in the ascending order.

Consider the following test code:

print("Merging [1,2,3],[4,3,2] :", merge\_lists([1,2,3],[4,3,2]))

print("Merging [3,2,1],[2,6,4,10,4] :", merge\_lists([3,2,1],[2,6,4,10,4]))

print("Merging [3,1,1],[] :", merge\_lists([3,1,1],[]))

print("Merging [],[9,7,2,7] :", merge\_lists([],[9,7,2,7]))

Below is the sample output when **lab5\_5** is run for the test code shown above

|  |
| --- |
| D:\is111\lab5>**python lab5\_5.py**  Merging [1,2,3],[4,3,2] : [1, 2, 3, 4]  Merging [3,2,1],[2,6,4,10,4] : [1, 2, 3, 4, 6, 10]  Merging [3,1,1],[] : [1, 3]  Merging [],[9,7,2,7] : [2, 7, 9] |

To submit**: lab5\_5.py**

1. In a file named **lab5\_6.py** copy the following code. Complete the function called get\_user\_info that takes a list of email addresses and returns a list of tuples containing user id, and domain. This exercise is adapted from the book “The Practice of Computing Using Python” Page 416, Exercise 45. The domain is the portion following @ sign in the email address.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19 | #write your functions get\_user\_info and print\_users here  name\_list = ["shaun.chew.2010@sis.smu.edu.sg","sitongchen.2011@economics.smu.edu.sg"]  users\_tuple = get\_user\_info(name\_list)    print("Users and their domain" )  print\_users(users\_tuple) |

Here is the expected output when **lab5\_6** is run:

|  |
| --- |
| D:\is111\lab5>**python lab5\_6.py**  Users and their domain  ('shaun.chew.2010', 'sis.smu.edu.sg')  ('sitongchen.2011', 'economics.smu.edu.sg') |

To submit**: lab5\_6.py**

1. \*In mathematics, a transpose of a matrix is a new matrix whose rows are the columns of the original.

If the matrix A is

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

The transpose of the matrix is (AT )

|  |  |  |
| --- | --- | --- |
| 1 | 4 | 7 |
| 2 | 5 | 8 |
| 3 | 6 | 9 |

In a file named **lab5\_7.py** write a function called transpose that takes a 2D- list of numbers representing a square matrix (i.e. number of columns and rows are equal) and returns a transposed list. The original list should not be changed.

For the example shown above,

the list representing the original would be [[1,2,3],[4,5,6],[7,8,9]]

and the transposed list would be [[1,4,7],[2,5,8],[3,6,9]]

Consider the following test code:

original = [[1,2],[3,4]]

print ("Original :", original)

print ("Transposed:", transpose(original) )

original = [[1,2,3],[4,5,6],[7,8,9]]

print ("Original :", original )

print ("Transposed:", transpose(original) )

Below is the sample output when **lab5\_7** is run (for the test code shown above):

|  |
| --- |
| D:\is111\lab5>**python lab5\_7.py**  Original : [[1, 2], [3, 4]]  Transposed: [[1, 3], [2, 4]]  Original : [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  Transposed: [[1, 4, 7], [2, 5, 8], [3, 6, 9]] |

**Note**: Try this code in the python shell

>>> list1 = [[1,2],[3,4]]

>>> list2 = list1.copy()

>>> list1[0][0] = 9

>>> list1

[[9, 2], [3, 4]]

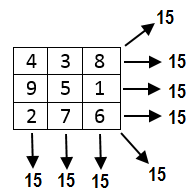
>>> list2

[[9, 2], [3, 4]]

We see from the above example that although we made a copy of list1, before making changes to list1, both list1 and list2 are changed. In order to make a proper copy of the 2d list, you could make use of copy module’s deepcopy() method. This method makes a proper copy of the 2d list.

To submit**: lab5\_7.py**

1. \*In a magic square, every row, column and diagonal add up to the same total. Here is a 3 by 3 magic square. The numbers 1 to 9 are placed in the grid such that no number is repeated and the sum of three digits column-wise, row-wise and diagonal-wise is equal to 15



The elements of the above magic square can be represented using a 2-dimensional list. The above square is represented by [[4,3,8],[9,5,1],[2,7,6]].

The following lists represent magic squares

[4,3,8],[9,5,1],[2,7,6] ,

[4,9,2],[3,5,7],[8,1,6] , [6,1,8],[7,5,3],[2,9,4]

while the following do not:

[5,5,5],[5,5,5],[5,5,5] ,

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

Write a function is\_magic\_square() in a file named **lab5\_8.py** that takes in a 2D list and returns True if the list contains all digits between 1 to 9 just once and it forms a magic square, False otherwise. You can assume that a proper 3 by 3 - 2D list will be sent to the function.

**Note**: We suggest that you design the program to contain another function that checks if only numbers from 1 to 9 are present in the list and make use of that function in is\_magic\_square.

Here is the expected output when **lab5\_8** is run (with the test case shown above) :

|  |
| --- |
| D:\is111\lab5>**python lab5\_8.py**  [[4,3,8],[9,5,1],[2,7,6]] magic square? : True  [[4,9,2],[3,5,7],[8,1,6]] magic square? : True  [[6,1,8],[7,5,3],[2,9,4]] magic square? : True  [[5,5,5],[5,5,5],[5,5,5]] magic square? : False  [[2,6,7],[9,5,1],[4,3,8]] magic square? : False  [[10,4,1],[1,5,9],[4,6,5]] magic square?: False |

To submit**: lab5\_8.py**