Question: 1

```
def find_average(student):
        roll, name, marks = student
        total = 0
        for mark in marks:
            total += mark
        avg = total / len (marks)
        print("Roll No: ", roll, "Name, "Average :", avg)


  stud1 = (1, "rex", (60, 85, 70))

  find_average (stud1)

  stud2 = (2, "rex", (80, 75, 90))

  find_average (stud2)
```

Question: 2

```
   List = []
   for i in range (7):
        c = float(input ("Enter your weight :"))
        List. append (c)
   print("\n \n First day weight", list [0])
   print ("\n Last day weight", List [-1])
   print ("\n Highest weight is", max (list))
   print ("\n Lowest weight is", min (list))
   print ("\n Average of weight is", round (sum(list)/
            len(list)))
```

Problem Solving Using Python and R Lab

Lab5. List Processing in Python

Question1. Write a function **find_average(student)** that takes student tuple as input and print student rollno, name, marks and average marks as output.

Test Cases:

1. stud1 = (1, "rex", 60, 85, 70)
   find_average(stud1)

Modify the above function **find_average(student)** so that it processes a tuple of tuples.

2. stud2 = (2, "rex", (80, 75, 90))
   find_average(stud2)

Question2. Write a weight management program that prompts the user to enter in 7 days of their body weight values as float numbers. Store them in list.
Then print first day weight, last day weight, 4th day weight, highest weight, lowest weight and average weight.
Finally, print if average weight < lowest weight, then print "Your weight management is excellent". Otherwise print "Your weight management is not good. Please take care of your diet".

Question:2

```
 Firstday = list [0]
 Lastday  = list [-1]
 avg = round (sum(list)/ len(list))
 low_weight = min(list)
 if (avg < low_weight):
        print("\n Your weight management is excellent")
else:
     print("\n Your weight management is not good.
     please take care of your diet")
```

Question 3:

```
def lastN(lst, n):
    final_list = []

    for i in range(n):
        max1 = 0

        for j in lst:
            if j > max1:
                max1 = j;
        lst.remove(max1);
        final_list.append(max1)
    print("\nlargest numbers:", final_list)

b = int(input("How many number want to enter?: "))

lst = [ ]
for i in range(b):
        c = int(input("\n Enter a number: "))
        lst.append(c)
    n = int(input("\nHow many largest numbers you want
            to find?: "))

lastN(lst, n)
```

Question 4:

```
def front_x(words):
        x list = []
        alist = []

        for word in words:
            if word.startswith("x"):
                xlist.append(word)
            else:
                alist.append(word)
        return sorted(xlist) + sorted(alist)
```

Question3. Write a function **lastN(lst, n)** that takes a list of integers and **n** and returns **n** largest numbers.

```
How many numbers you want to enter?: 6
Enter a number: 12
Enter a number: 32
Enter a number: 10
Enter a number: 9
Enter a number: 52
Enter a number: 45
How many largest numbers you want to find?: 3
Largest numbers are: 52, 45, 32
```

Question4. Given a list of strings, return a list with the strings in sorted order, except group all the strings that begin with 'x' first. Hint: this can be done by making 2 lists and sorting each of them before combining them.

Test Cases:

1. Input: ['mix', 'xyz', 'apple', 'xanadu', 'aardvark']
   Output: ['xanadu', 'xyz', 'aardvark', 'apple', 'mix']
2. Input: ['ccc','bbb','aaa','xcc','xaa']
   Output: ['xaa','xcc','aaa','bbb','ccc']
3. Input: ['bbb','ccc','axx','xzz','xaa']
   Output: ['xaa','xzz','axx','bbb','ccc']

Question 4:

```
words = ['min', 'xyz', 'apple', 'xanadu', 'aardvark']
front_x(words)
words = ['ccc', 'bbb', 'aaa', 'xcc', 'xaa']
front_x(words)
words = ['bbb', 'ccc', 'axx', 'xzz', 'xaa']
front_x(words)
```

Question:5

```
def last(n): return n[-1]

 def sort_list_last(tuples):
       return Sorted (tuples, Key = last)
 print ("Test Case:1 \n\t Input: [(1,7), (1,3), (3,4,5), (2,2)]
        \n\output:", sortec list_ last ([(1,7), (1,3), (3,4,5), (2,2)]))
 Print ("Test Case:2 \n\t Input: [(1,3), (3,2), (2,1)] \n \toutput:"
        ,sort_List_last ([(1,3), (3,2), (2,1)]))
  print ("Test Case:3 \n\t Input: [(2,3), (1,2), (3,1)] \n\toutputs"
        , sort_List_last ([(2,3), (1,2), (3,1)]))
```

Question 6:

```
        def first (num):
              for j in num:
                  a, b, c = j
                  print (a)
         def sort_first(num):
                new_list = sorted (num)
                return new_list

         def List (num):
                return sorted (num, key = None, reverse = 0)

          def middle (num):
                for j in num:
                    a, b, c = j
                    print (b)
```

Question5. Develop a function **sort_last()**. Given a list of non-empty tuples, return a list sorted in increasing order by the last element in each tuple. Hint: use a custom key= function to extract the last element form each tuple.

Test Cases:

1. Input: [(1, 7), (1, 3), (3, 4, 5), (2, 2)]
   Output: [(2, 2), (1, 3), (3, 4, 5), (1, 7)]
2. Input: [(1,3),(3,2),(2,1)]
   Output: [(2,1),(3,2),(1,3)]
3. Input: [(2,3),(1,2),(3,1)]
   Output: [(3,1),(1,2),(2,3)]

Question6. Other String Functions

a) Define a function **first()** that receives a tuple and returns its first element
b) Define a function **sort_first()** that receives a list of tuples and returns the sorted
c) Print lists in sorted order
d) Define a function **middle()** that receives a a tuple and returns its middle element
e) Define a functino **sort_middle**() that receives a list of tuples and returns it sorted using the key middle
f) Print the list [(1,2,3), (2,1,4), (10,7,15), (20,4,50), (30, 6, 40)] in sorted order. Output should be: [(2, 1, 4), (1, 2, 3), (20, 4, 50), (30, 6, 40), (10, 7, 15)]

Question 6:

```
def sort_middle (num):
    return sorted (num, key = lambda mid: mid[1])
num = [(1,2,3), (2,1,4), (10,7,15), (20,4,50), (30,6,40)]

⇒ first(num)
⇒ sort_first(num)
⇒ middle (num)
⇒ sort_middle (num)
```

Question7. Develop a function **remove_adjacent().** Given a list of numbers, return a list where all adjacent same elements have been reduced to a single element. You may create a new list or modify the passed in list.

Test Cases:

1. Input: [1, 2, 2, 3] and output: [1, 2, 3]
2. Input: [2, 2, 3, 3, 3] and output: [2, 3]
3. Input: [ ]. Output: [ ].
4. Input: [2,5,5,6,6,7]
   Output: [2,5,6,7]
5. Input: [6,7,7,8,9,9]
   Output: [6,7,8,9]

Question 7:

```
def remove_adjacent (lst):
    a = [ ]
    for item in lst:
        if len (a):
            if a[-1] != item:
                a.append (item)
        else:
            a.append (item)
    return a

remove_adjacent ([1,2,2,3])
o/p: [1,2,3]
remove_adjacent ([2,2,3,3,3]
o/p: [2,3]
```

Question8. Write a function **verbing()**. Given a string, if its length is at least 3, add 'ing' to its end. Unless it already ends in 'ing', in which case add 'ly' instead. If the string length is less than 3, leave it unchanged. Return the resulting string. So 'hail' yields: hailing; 'swimming' yields: swimmingly; 'do' yields: do.

Question8:

```
def verbing (str):
        length = len(str)
        if length >2:
            if str[-3:] =='ing':
                str +='ly'
            else:
                str += 'ing'
        return str
```

Question9. Develop a function **not_bad()**. Given a string, find the first appearance of the substring 'not' and 'bad'. If the 'bad' follows the 'not', replace the whole 'not'...'bad' substring with 'good'.
Return the resulting string. So 'This dinner is not that bad!' yields: This dinner is good!

Question 9:

```
def not_bad (s):
        snot = s.find('not')
        sbad = s.find('bad')
        if sbad >snot:
            s =s.replace(s[snot:(sbad+3)],'good')
        return s
```

not_bad ("This dinner is not really that bad!")

o/p: " This dinner is good!"