15-110 Refresher Session : Week 9

No Calculators, only Brains !!

1. **Act like a Computer**

```
def for_mystery(L) :
    sol = []
    for x in range(len(L)) :
        sol.append(x)
    for x in L :
        sol.append(x)
    return sol

def if_mystery(L , x) :
    if x in L :
        return True
    else :
        return False

def while_mystery(L, x) :
    while x in L :
        L.remove(x)
    return L

def mystery(L) :
    for_sol = for_mystery(L)
    if_sol = if_mystery(L , for_sol[0])
    while_sol = while_mystery(L , if_sol)
    print(while_sol)
```

```
mystery([2, True,  4, 0, True])
```

What would the output be for the following function calls ?

(a) `mystery([5, True, 4, 0, True])`

(b) `mystery([2, False, 4 ,True])`

2. **Act like a Programmer**
Sam learnt the quadratic formula in Math class and wanted to use Python to aid his learning so that he could cross check the solutions. He has to solve a lot of problems from the text book as a homework assignment. He creates a list of lists with each inner list containing three elements - `a`, `b`, `c` and aims to return a list of tuples with each tuple containing two elements - the solutions of the quadratic equation.

In this problem, you have to help sam by implementing a function `solveQuadratic(L)` which takes a list of input values with each list containing three elements - `a`, `b`, `c` and returns a list of tuples with each tuple consisting of two elements - the solutions of the quadratic equation rounded to 2 decimal places. If there are no solutions, the solutions are stated to be `False`. We can assume that Sam might make errors and forget one or more of the elements in inputs, and if he does, that input should be taken as 1 by default. Review the example below.

```
solveQuadratic([[2 , -3 , -3], [2 , 3] , [0 , 10  ,2 ] , [-9]])
returns [(2.19, -0.69), (-0.5, -1.0), (False,  False), (-0.28, 0.39)]
```

Hint : You can use helper functions to solve the equations separately and return them

3. **Act like a Computer**

```
def mystery_1(L):
    L = L * 2
    return L

def mystery_2(L):
    L.sort()
    L.append(64)
    return L

def mystery_3(a) :
    a += 20
    return a

def mystery():
    global L
    newList_1 = mystery_1(L)
    print(L, newList_1)
    newList_2 = mystery_2(L)
    print(L, newList_2)
    newNum = mystery_3(L[-1])
    print(newNum, L)

L = [100 , 430 , 70]
mystery()
print(L)
```

Act like a computer and write down the output for the code given above.

4. **Act like a Programmer**
Fred attempted an unknown number of exams and wanted to review how his scores were distributed across each question so that he could prepare better for the next attempt. Your task is to help him by implementing a function. Write a function `examAnalysis(L, maxScore)` which takes a list of lists `L` with each inner list containing the scores to each question and an integer `maxScore` - the maximum score of each question. Assuming that every question has equal weightage, he wants to create a list of tuples with the following data :

(a) his highest scoring question in each exam - if there is more than one, you can return either

(b) his lowest scoring question in each exam - if there is more than one, you can return either 0verage of each exam

The list should also contain an additional tuple with the averages of each question at the end. Review the example below :

```
examAnalysis([(12, 13, 14), (2, 1, 0), (5, 4 ,7) , (6, 0 ,3) ], 15)
returns [(3, 1, 13.0), (1, 3, 1.0), (3, 2, 5.33), (1, 2, 3.0), (42.0, 30.0, 40.0)]
```