This coding assignment is designed to help you get up to speed with coding in Python. The first one concerns functional programming and the second concerns OOP.

Working in group is very much encouraged. But please make sure that a group consists of no more than three people. If you can't find any teammate and end up working alone, you will receive a little credit for trying this alone so that everyone, working alone or not, tends to work in a level playing field.

Please save your solutions in a file that ends with '.py' so that it is a proper executable Python file. Please indicate your group members names in a separate text file so that I know who you work with. Then please make a compressed folder that has these files in it.

Good luck!

Q1. (50 points)

The following sequence is formed using words and numbers:
(1) The first number is 1
(2) In the first number, there was one 1 so the second number in the sequence is 11
(3) In the second number, there were two 1s so the third number in the sequence is 21
(4) In the previous number, there was one 2 and there was one q, so the fourth number is 1211
(5) In the previous number, there was one 1, one 2, and two 1s, so the fifth number is 111221...
This sequence can continue infinitely.
Write a function that, given a staring number m and an integer n, returns the next n numbers in order.
So for example, given a starting number of 1 and an integer 4, the sequence returned should be [11, 21, 1211, 111221]
Or, given a starting number of 11 and an integer 3, the sequence returned should be [21, 1211, 111221]

Q2. (50 points)

Motivation

This exercise focuses on the very basic aspect of OOP: encapsulation. Therefore, there is no need for you to be concerned about inheritance or polymorphism or other advanced concepts of OOP.

We have spent the past two weeks learning NumPy. An NumPy array is a list that consists of data of the same data type. But it also has a lot of nice methods or operations associated with it, which makes the life of a data scientist a lot easier.

For instance, you have array A ([1, 2, 3, 4]) and array B ([3, 4, 5, 6]). You can easily do an operation where each element in A and that in B can add up and collectively make a new array C ([4, 6, 8, 10]). With NumPy this can be easily done using the '+' operator.

You may also turn array A and B into a 2D array, respectively, by using the 'reshape' method. In other words, you may do this as follows.

```
arr1_2d = np.array([1,2,3,4]).reshape((1,4))
arr2_2d = np.array([3,4,5,6]).reshape((1,4))
```

Or, you may turn each of them into a 2D array of the shape (2,2). The point is given a simple (not nested) list, you can easily make a 2D array.
A 2D array mimics a matrix. To perform multiplication of two matrices (2D arrays), you need to use np.dot(matA, matB) where matA and matB are the two matrices multiplied. Of course, to legitimately multiply matA by matB, their dimensionality and shape need to match: the number of the columns of matA has to be equal to the number of the rows of matB.

The problem

So let us try to create a few customary classes that can deal with the kinds of operations we just mentioned here.

We need two classes to accomplish this goal. We need a class from which we can create a NumPy array like object. Let's call this class 'MyNumMatrix'. I have created this class for you:

```
class MyNumMatrix:
    def __init__(self, values, shape):
        self.values = values
        self.shape = shape

    def __str__(self):
        return "values:{self.values} shape {self.shape}".format(self=self)

#see here for more info on how repr and str work in Python.
#https://www.educative.io/answers/what-is-the-str-method-in-python

    def __repr__(self):
        return "values:{self.values} shape {self.shape}".format(self=self)
```

Notice the properties of this class: values and shape. Values is a list of numbers given and shape is a tuple of 2 numbers which indicate the rows number and columns number of the 2D array (matrix) you want to create. So if you have mat1_2d = MyNumMatrix([1, 2 ,3 ,4], shape=(2,2)), then mat1_2d will be a matrix of 2 rows and 2 columns where the first row consists of 1 and 2 and the second row consists of 3 and 4.

One thing worth noting: '__str__' and '__repr__' are used to customize the 'look' of a customary object. In Python, the 'look' of a customary object is not informative at all such that you need to overwrite the 'print' function/method. This way, when you print an object, its properties will show up.

Now please finish creating a second class that, given two matrices MatA and MatB (from MyNumMatrix class), can do the '+' operation and the np.dot( )operation.

```
class MyNumRule:

    def __init__(self, matrices):
        self.matrices = matrices #matrices is a list of two matrices of the
#MyNumMatrix class.

    def sum(self):
        #this method does what '+' operation does and returns the sum of two
#matrices and that sum is a matrix of the MyNumMatrix class.
```

def multiply(self):
    #this method does what ‘np.dot()’ does and returns the product of two #matrices and that product is a matrix of the MyNumMatrix class.

The following code serves as the test cases. You design is expected to run through these test cases and return correct values.

```
mat1 = MyNumMatrix([1,2,3,4], shape=(1,4))
mat2 = MyNumMatrix([3,4,5,6], shape=(1,4))
opera1 = MyNumRule([mat1, mat2])
opera1.sum()

mat1_2d = MyNumMatrix([1, 2 ,3 ,4], shape=(2,2))
mat2_2d = MyNumMatrix([3,4,5,6], shape=(2,2))
opera2 = MyNumRule([mat1_2d, mat2_2d])
opera2.sum()
opera2.multiply()
```