Revision Class 3 (Dated: 16-Sep-2022)

Note: Apologies in advance, if some code may have some typing or other mistakes, due to lack of time (working on exams of your seniors), I couldn't run these codes (for verification). You may inform through your CR, for any correction required.

1. **[Question from Practice 9]** Given a 2D list of integers, print average of every 3x3 sub-list inside the 2D list

   Explanation: Consider the blue box, having 9, 9, 5 in first row, 4, 7, 3 in second row and 10, 4, 4 in last row. The average of these values is (9+9+5+4+7+3+10+4+4) = 55 / 9 = 6 (Integer Division). In both left side and right side values in blue shade are corresponding. Means, in right side block bold, blue shaded 6 is average of numbers in left side block blue shaded.

   Similarly, next block on the left is shown in orange color and corresponding average value 5 is shown in orange color in the right side block. Anyhow, you have to write a general code for any square 2D list.

9	9	5	9	4
4	7	3	7	5
10	4	4	2	4
5	2	10	1	9
8	6	10	5	6

6	**5**	**4**
5	**4**	**5**
6	**4**	**5**

Method 1:

```
def print_average_3_by_3(a):
    for i in range(len(a) - 2):
        for j in range(len(a[i])-2):  #len(a[i]) is more generic, applicable for both square
                                      # and other rectangular 2D lists
            sum = 0
            for m in range (i, i+3):
                for n in range (j, j+3):
                    sum += a[m][n]
            print(f'{sum//9:2d}', end=' ')
        print ()
```

Method 2: [Using another function]

```
def get_average_3_3(a, i, j):
    sum = 0
    for m in range (i, i+3):
        for n in range (j, j+3):
            sum += a[m][n]
    return sum // 9

def print_average_3_by_3_using_another_function(a):
    for i in range(len(a) - 2):
        for j in range(len(a[i])-2):
            print(f'{get_average_3_3(a, i, j):2d}', end=' ')
        print ()
```

2. **[Question from Lab 6]** Write following function for a 1D list of numbers:

Print list in matrix/ row-column form (Function has 3 parameters list, rows & columns). It is given that list has at least rows x columns elements. See example:

Consider: 1 2 3 4 5 6 7 8 9 10 11 12
* Print in 3 rows and 4 columns

```
1  2  3  4
5  6  7  8
9 10 11 12
```

* Print in 2 rows and 6 columns

```
1  2  3  4   5   6
7  8  9  10  11  12
```

Method 1:

```python
def print_row_column_form1(arr, rows, columns):
    c = 0
    for element in arr:
        print (f'{element:2d} ', end='')
        c += 1
        if c ==  columns:
            c = 0
            print()
```

Method 2: [Very much similar to first method]

```python
def print_row_column_form2(arr, rows, columns):
    c = 0
    for i in range(len(arr)):
        print (f'{arr[i]:2d} ', end='')
        c += 1
        if c ==  columns:
            c = 0
            print()
```

Method 3: [Using nested loop, without if-else]

```python
def print_row_column_form3(arr, rows, columns):
    c = 0
    for i in range(rows):
        for j in range(columns):
            print (f'{arr[c]:2d} ', end='')
            c += 1
        print()
```

Method 4: [Using mapping function]

```python
def print_row_column_form4(arr, rows, columns):
    for i in range(rows):
        for j in range(columns):
      #i * columns + j is a mapping function from 2D (using i & j) to 1D
            print (f'{arr[i*columns+j]:2d} ', end='')
        print()
```

3. **[Question from Lab 4]** Declare an empty list. Append 20 **unique** values in the list at random in range 10-99. Display values in a single line. Try your best to write the code to generate unique values, however, if you fail to do so in ten minutes, leave your unique generation code in comments and initialize list with unique values in range 10-99. One sample list is:

```
[30, 33, 60, 47, 36, 64, 27, 57, 25, 83, 24, 89, 95, 94, 53, 97, 75, 35, 11, 38]
```

Take a new empty list. Copy values from old list into new list. Shuffle new list 100 times that is run a loop of 100 times, select two random indexes within the range of list and swap values at both index. See the sample new list after shuffle, match it with old list to understand the shuffle:

```
[57, 97, 83, 30, 89, 33, 75, 11, 47, 94, 38, 60, 95, 36, 27, 24, 25, 35, 64, 53]
```

Next, match every element of the old list in the new list and print element and its index in new list. See sample output again:

```
30 exists at index 3 in second list

33 exists at index 5 in second list
```

```
...
```

Finally, find distance of each element in the old list with same element in the new list. Use **abs** function to find absolute distance. Find and print the maximum distance between two values:

```
Maximum distance between two elements is: 14
```

Node down the element 97 at index 15 in first list and it is at index 1 in second list, therefore distance is 14, which is maximum distance between any two elements.

```
import random as r

def not_exist(a, element):
    for list_element in a:
        if list_element == element:
            return False
    return True

def append_20_unique_values(a):
    i = 0
    while i < 20:
        val = r.randint(10,99)
        if not_exist(a,val):
            a.append(val)
            i += 1

def shuffle(a, times):
    for i in range(times):
        m = r.randint(0,19)
        n = r.randint(0,19)
        a[m], a[n] = a[n], a[m]

def compare(a, b):
    max_d = 0
    for i in range(len(a)):
        for j in range(len(a)):
            if a[i] == b[j]:
                d = abs(i - j)
                print (f'{a[i]} exists at index {j} in second list')
                if max_d < d:
                    max_d = d
    print(f'Maximum distance between two elements is: {max_d}')

def main():
    a =[]
    append_20_unique_values(a)
    print(a)
    b=[element for element in a]
    shuffle(b, 100)
    print(b)
    compare(a, b)
```

4. Given a 1D list. write function to rotate n times, see example for understanding:

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

[10, 1, 2, 3, 4, 5, 6, 7, 8, 9]   All elements will move to 1 right; where last element will move to start of the list. Similarly, after 3 rotations, the list will be: [8, 9, 10, 1, 2, 3, 4, 5, 6, 7]

a. Rotate elements of first list into second list without modification of first list

Method 1: [Using two loops]

```
import random as r

def rotate1(a, times):
    b=[]
    for i in range(len(a)-times, len(a)):
        b.append(a[i])
    for i in range(len(a)-times):
        b.append(a[i])
    return b

def main():
    a =[i for i in range(10)]
    print(a)
    print(rotate1(a,3))
```

Method 2: [Using one loop and % operator]

```
def rotate2(a, times):
    b=[]
    i = len(a)-times
    while len(b)<len(a) :
        b.append(a[i])
        i = ( i +1 ) % len(a)
    return b
```

b. Rotate elements of list within in the list

Method 1: [may consume more memory]

```
def rotate_within_list(a, times):
    length =len(a)
    for i in range(length-times):
        a.append(a[i])
    for i in range(length-times):
        a.remove(a[0])
    return a
```

After rotation, some starting elements will move to the end of list, therefore first loop is adding such elements at the end of list. Second loop will remove them from start of the list.

Method 2: [memory efficient, we are trying our best to use minimal extra memory]

```
def rotate_within_list1(a, times):
    length =len(a)
    if times < length - times:
        temp=[a[i] for i in range(length-times, length)]
        for i in range(length-1, times-1, -1):
            a[i] = a[i-times]
        for i in range(times):
            a[i] = temp[i]
    else:
        temp = [a[i] for i in range(length-times)]
        j = 0
        for i in range(length-times, length):
            a[j] = a[i]
            j += 1
        for i in range(length-times):
            a[times+i] = temp[i]
    return a
```

We will find, how many elements to be moved to the temporary list. If the rotations are lesser than length of the list, this means there are lesser elements on the right side of the list, to be moved to left side (i.e. at the start of the list later on). We have moved those smaller number of elements into a temporary array. Then, we have moved our list values towards right side starting from right moving towards left. Therefore, first we will move value to last index, then second last index and so on. Finally, we have added the elements from temporary list at the start of our original list.

Similarly, otherwise, we have smaller elements on the left hand side, to be moved to the end of the list later on. We have copied them into temporary array. We have moved remaining elements to the left starting from index 0 to onwards from left to right. At the end, we have added elements from temporary array at the end of our original list.