Two-dimensional lists (arrays)

1. Nested lists: processing and printing

In real-world Often tasks have to store rectangular data table. [say more on this!] Such tables are called *matrices* or two-dimensional arrays. In Python any table can be represented as a list of lists (a list, where each element is in turn a list). For example, here's the program that creates a numerical table with two rows and three columns, and then makes some manipulations with it:

```
a = [[1, 2, 3], [4, 5, 6]]

print(a[0])

print(a[1])

b = a[0]

print(b)

print(a[0][2])

a[0][1] = 7

print(a)

print(b)

b[2] = 9

print(a[0])

print(b)
```

The first element of `a` here — `a[0]` — is a list of numbers `[1, 2, 3]`. The first element of *this* new list is `a[0][0] == 1`; moreover, `a[0][1] == 2`, `a[0][2] == 3`, `a[1][0] == 4`, `a[1][1] == 5`, `a[1][2] == 6`.

To process 2-dimensional array, you typically use nested loops. The first loop iterates through the row number, the second loop runs through the elements inside of a row. For example, that's how you display two-dimensional numerical list on the screen line by line, separating the numbers with spaces:

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

for i in range(len(a)):

    for j in range(len(a[i])):

        print(a[i][j], end=' ')

    print()
```

We have already tried to explain that a for-loop variable in Python can iterate not only over a `range()`, but generally over all the elements of any sequence. Sequences in Python are lists and strings (and some other objects that we haven't met yet). Look how you can print a two-dimensional array, using this handy feature of loop `for`:

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

for row in a:

    for elem in row:

        print(elem, end=' ')

    print()
```

Naturally, to output a single line you can use method `join()`:

```
for row in a:

    print(' '.join([str(elem) for elem in row]))
```

This is how you can use 2 nested loops to calculate the sum of all the numbers in the 2-dimensional list:

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

s = 0

for i in range(len(a)):

    for j in range(len(a[i])):

        s += a[i][j]

print(s)
```

Or the same with iterating by elements, not by the variables `i` and `j`:

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

s = 0

for row in a:

    for elem in row:

        s += elem

print(s)
```

Nested lists: creating

Suppose that two numbers are given: the number of rows of `n` and the number of columns `m`. You must create a list of size `n`×`m`, filled with, say, zeros.

The obvious solution appears to be wrong:

```
a = [[0] * m] * n
```

This can be easily seen if you set the value of `a[0][0]` to `5`, and then print the value of `a[1][0]` — it will also be equal to 5. The reason is, `[0] * m` returns just a *reference* to a list of `m` zeros, but not a list. The subsequent repeating of this element creates a list of `n` items that all reference to the same list (just as well as the operation `b = a` for lists does not create the new list), so all rows in the resulting list are actually the same string.

Using our visualizer, keep track of the id of lists. If two lists have the same id number, it's actually the same list in memory.

```
n = 3

m = 4

a = [[0] * m] * n

a[0][0] = 5

print(a[1][0])
```

Thus, a two-dimensional list cannot be created simply by repeating a string. What to do?..

A possible way: you can create a list of `n` elements (say, of `n` zeros) and then make each of the elements a link to another one-dimensional list of `m` elements:

```
n = 3

m = 4

a = [0] * n

for i in range(n):

    a[i] = [0] * m
```

Another (but similar) way: create an empty list and then `append` a new element to it `n` times (this element should be a list of length `m`):

```
n = 3

m = 4

a = []

for i in range(n):

    a.append([0] * m)
```

But the easiest way is to use generator, creating a list of `n` elements, each of which is a list of `m` zeros:

```
n = 3

m = 4

a = [[0] * m for i in range(n)]
```

In this case each element is created independently from the others. The list `[0] * m` is `n` times consructed as the new one, and no copying of references occurs.

3. How do you input a two-dimensional array?

Say, a program takes on input two-dimensional array in the form of `n` rows, each of which contains `m`numbers separated by spaces. How do you force the program to read it? An example of how you can do it:

```
# the first line of input is the number of rows of the array

n = int(input())

a = []

for i in range(n):

    a.append([int(j) for j in input().split()])
```

Or, without using sophisticated nested calls:

```
# the first line of input is the number of rows of the array

n = int(input())

a = []

for i in range(n):

    row = input().split()

    for i in range(len(row)):

        row[i] = int(row[i])

    a.append(row)
```

You can do the same with generators:

```
# the first line of input is the number of rows of the array

n = int(input())

a = [[int(j) for j in input().split()] for i in range(n)]
```

4. Processing a two-dimensional array: an example

Suppose you are given a square array (an array of `n` rows and `n` columns). And suppose you have to set elements of the main diagonal equal to 1 (that is, those elements `a[i][j]` for which `i==j`), to set elements above than that diagonal equal to 0, and to set elements below that diagonal equal to 2. That is, you need to produce such an array (example for `n==4`):

```
1 0 0 0

2 1 0 0
```

2 2 1 0

2 2 2 1

(In this case you can do it manually by setting `a[0][0] = 1`, `a[0][1] = 0` and so on, but you won't do it manually for arrays of 100 rows and 100 columns, which are often the case.)

We are eager to show you several ways of solving this problem. First, note that elements that lie above the main diagonal – are elements `a[i][j]` for which `i<j`, and that for elements below the main diagonal `i>j`. Thus, we can compare the values `i` and `j`, which determines the value of `a[i][j]`. We get the following algorithm:

```
n = 4

a = [[0] * n for i in range(n)]

for i in range(n):

    for j in range(n):

        if i < j:

            a[i][j] = 0

        elif i > j:

            a[i][j] = 2

        else:

            a[i][j] = 1

for row in a:

    print(' '.join([str(elem) for elem in row]))
```

This algorithm is slow: it uses two loops and for each pair `(i,j)` executes one or two `if` instructions. If we complicate the algorithm, we will be able to do it without a conditional statement.

First, fill the main diagonal, for which we will need one loop:

```
for i in range(n):

    a[i][i] = 1
```

Then fill with zeros all the elements above the main diagonal. To make this, for each row with the number `i`you need to assign a value to `a[i][j]` for `j`=`i+1`, ..., `n-1`. To do that, you need nested loops:

```
for i in range(n):

    for j in range(i + 1, n):

        a[i][j] = 0
```

By analogy, for `j`=`0`, ..., `i-1` set the elements `a[i][j]` equal to `2`:

```
for i in range(n):

    for j in range(0, i):
```

```
        a[i][j] = 2
```

You can combine all this code and receive another solution:

```
n = 4
a = [[0] * n for i in range(n)]
for i in range(n):
    for j in range(0, i):
        a[i][j] = 2
    a[i][i] = 1
    for j in range(i + 1, n):
        a[i][j] = 0
for row in a:
    print(' '.join([str(elem) for elem in row]))
```

Here's another solution, which repeats lists to build the next rows of the list. The `i`-th line of the list consists of `i` numbers `2`, followed by one integer `1`, followed by `n-i-1` zeros:

```
n = 4
a = [0] * n
for i in range(n):
    a[i] = [2] * i + [1] + [0] * (n - i - 1)
for row in a:
    print(' '.join([str(elem) for elem in row]))
```

As usual, you can replace the loop with the generator:

```
n = 4
a = [0] * n
a = [[2] * i + [1] + [0] * (n - i - 1) for i in range(n)]
for row in a:
    print(' '.join([str(elem) for elem in row]))
```

5. Two-dimensional arrays: nested generators

You can use nested generators to create two-dimensional arrays, placing the generator of the list which is a string, inside the generator of all the strings. Recall that you can create a list of `n` rows and `m` columns using the generator (which creates a list of `n` elements, where each element is a list of `m` zeros):

```
[[0] * m for i in range(n)]
```

But the internal list can also be created using, for example, such generator: `[0 for j in range(m)]`. Nesting one generator into another, we obtain

```
[[0 for j in range(m)] for i in range(n)]
```

How is it related to our problem? The thing is, if the number 0 is replaced by some expression that depends on `i` (the line number) and `j` (the column number), you get the matrix filled according to some formula.

For example, suppose you need to initialize the following array (for convenience, extra spaces are added between items):

0 0 0 0 0 0

0 1 2 3 4 5

0 2 4 6 8 10

0 3 6 9 12 15

0 4 8 12 16 20

In this array there are `n = 5` rows, `m = 6` columns, and the element with row index `i` and column index `j`is calculated by the formula `a[i][j] = i * j`.

As always, you could use generator to create such an array:

```
[[i * j for j in range(m)] for i in range(n)]
```

Problem «Maximum» (Easy)

Statement

Given two integers representing the rows and columns $(m \times n)$(m×n), and subsequent mm rows of nn elements, find the index position of the maximum element and print two numbers representing the index $(i \times j)$(i×j) or the row number and the column number. If there exist multiple such elements in different rows, print the one with smaller row number. If there multiple such elements occur on the same row, output the smallest column number.

Your solution

```
m, n = [int(s) for s in input().split()]

a = [[int(j) for j in input().split()] for i in range(m)]

max_num = max(a[0])

min_j = n-1

max_i = m-1

if m == 1:

    for i in range(m):
```

```
        for j in range(n):
            if a[i][j]==max_num and j<min_j:
                min_j = j
else:
    for i in range(m):
        for j in range(n):
            if a[i][j] > max_num:
                max_num = a[i][j]
                max_i = i
                min_j = j
print(max_i, min_j)
```

Suggested solution

```
n, m = [int(i) for i in input().split()]
a = [[int(j) for j in input().split()] for i in range(n)]
best_i, best_j = 0, 0
curr_max = a[0][0]
for i in range(n):
    for j in range(m):
        if a[i][j] > curr_max:
            curr_max = a[i][j]
            best_i, best_j = i, j
print(best_i, best_j)
```

Problem «Snowflake» (Easy)

Statement

Given an odd number integer nn, produce a two-dimensional array of size $(n \times n)$(n×n). Fill each element with a single character string of `"."` . Then fill the middle row, the middle column and the diagnals with the single character string of `"*"` (an image of a snow flake). Print the array elements in $(n \times n)$(n×n) rows and columns and separate the characters with a single space.

Your solution

```
n = int(input())
m = n
```

```
s = '*'
d = '.'
a = [[d] * m for i in range(n)]
a[n//2] = [s]*n
for i in range(n):
    a[i][i] = s
    a[i][n//2] = s
    for j in range(n-1,-1,-1):
        a[n-1-i][i] = s
for row in a:
    print(' '.join([str(elem) for elem in row]))
```

Suggested solution

```
n = int(input())
a = [['.'] * n for i in range(n)]
for i in range(n):
    a[i][i] = '*'
    a[n // 2][i] = '*'
    a[i][n // 2] = '*'
    a[i][n - i - 1] = '*'
for row in a:
    print(' '.join(row))
```

Problem «Chess board» (Medium)

Statement

Given two numbers nn and mm. Create a two-dimensional array of size $(n \times m)$(n×m) and populate it with the characters `"."` and `"*"` in a checkerboard pattern. The top left corner should have the character `"."` .

Your solution

```
n, m = [int(s) for s in input().split()]
s = '*'
d = '.'
```

```
a = [[d] * m for i in range(n)]
x=len(a)
if len(a) % 2 == 0:
    for i in range(n):
        if (i +1) % 2 != 0:
            for j in range(m):
                if (j+1) % 2 == 0:
                    a[i][j] = s
        else:
            for j in range(m):
                if (j+1) % 2 != 0:
                    a[i][j] = s
else:
    for i in range(n):
        if (i +1) % 2 != 0:
            for j in range(m):
                if (j+1) % 2 == 0:
                    a[i][j] = s
        else:
            for j in range(m):
                if (j+1) % 2 != 0:
                    a[i][j] = s
for row in a:
    print(' '.join([str(elem) for elem in row]))
```

Suggested solution

```
n, m = [int(i) for i in input().split()]
a = []
for i in range(n):
    a.append([])
    for j in range(m):
        if (i + j) % 2 == 0:
```

```
            a[i].append('.')
        else:
            a[i].append('*')
for row in a:
    print(' '.join(row))
```

Problem «The diagonal parallel to the main» (Medium)

Statement

Given an integer n, produce a two-dimensional array of size $(n \times n)$ and complete it according to the following rules, and print with a single space between characters:

- On the main diagonal write `0`.
- On the diagonals adjacent to the main, write `1`.
- On the next adjacent diagonals write `2` and so forth.

Print the elements of the resulting array.

Your solution

```
n = int(input())
a = [[0] * n for i in range(n)]
for i in range(n):
    for j in range(n):
        for b in range(1,n):
            if i == j + b or i == j - b:
                a[i][j] = b
for row in a:
    print(' '.join([str(elem) for elem in row]))
```

Suggested solution

```
n = int(input())
a = [[abs(i - j) for j in range(n)] for i in range(n)]
for row in a:
    print(' '.join([str(i) for i in row]))
```

Problem «Side diagonal» (Hard)

Statement

Given an integer nn, create a two-dimensional array of size $(n \times n)$(n×n) and populate it as follows, with spaces between each character:

- The positions on the minor diagonal (from the upper right to the lower left corner) receive `1` .
- The positions above this diagonal recieve `0` .
- The positions below the diagonal receive `2` .

Print the elements of the resulting array.

Your solution

```
n = int(input())

a = [[0] * n for i in range(n)]

for i in range(n):

   a[i] = [0] * (n - i - 1)+ [1] + [2] * i

for row in a:

   print(' '.join([str(i) for i in row]))
```

Suggested solution

```
n = int(input())

a = [[0] * n for i in range(n)]

for i in range(n):

   a[i][n - i - 1] = 1

for i in range(n):

   for j in range(n - i, n):

      a[i][j] = 2

for row in a:

   for elem in row:

      print(elem, end=' ')

   print()
```

Problem «Swap the columns» (Hard)

Statement

Given two positive integers m and n, m lines of n elements, giving an $m \times n$ matrix A, followed by two non-negative integers i and j less than n, swap columns i and j of A and print the result.

Write a function `swap_columns(a, i, j)` and call it to exchange the columns.

Your solution

```
m, n = [int(i) for i in input().split()]
a = [[int(j) for j in input().split()] for i in range(m)]
I, J = [int(i) for i in input().split()]
d = []
e = []
for i in range(m):
    for j in range(n):
        if j == J :
            d.append(a[i][j])
        if j == I :
            e.append(a[i][j])
for i in range(m):
    for j in range(n):
        if j == J:
            a[i][j] = e[i]
        if j == I:
            a[i][j] = d[i]
for row in a:
    print(' '.join([str(i) for i in row]))
```

Suggested solution

```
def swap_columns(a, i, j):
    for k in range(len(a)):
        a[k][i], a[k][j] = a[k][j], a[k][i]
```

```
n, m = [int(i) for i in input().split()]
a = [[int(j) for j in input().split()] for i in range(n)]
i, j = [int(i) for i in input().split()]
swap_columns(a, i, j)
print('\n'.join([' '.join([str(i) for i in row]) for row in a]))
```

Problem «Scale a matrix» (Easy)

Statement

Given two positive integers m and n, m lines of n elements, giving an $m \times n$ matrix A, followed by one integer c, multiply every entry of the matrix by c and print the result.

Your solution

```
n, m = [int(i) for i in input().split()]
a = [[int(j) for j in input().split()] for i in range(n)]
c = int(input())
for i in range(n):
    for j in range(m):
        a[i][j] *= c
print('\n'.join([' '.join([str(i) for i in row]) for row in a]))
```

Suggested solution

```
m, n = [int(k) for k in input().split()]
A = [[int(k) for k in input().split()] for i in range(m)]
c = int(input())
for i in range(m):
    for j in range(n):
        A[i][j] *= c

print('\n'.join([' '.join([str(k) for k in row]) for row in A]))
```

Problem «Multiply two matrices» (Easy)

Statement

Given three positive integers mm, nn and rr, mm lines of nn elements, giving an $m \times nm \times n$ matrix AA, and nn lines of rrelements, giving an $n \times rn \times r$ matrix BB, form the product matrix $ABAB$, which is the $m \times rm \times r$ matrix whose (i,k)(i,k)entry is the sum

$$A[i][1] * B[1][k] + \cdots + A[i][n] * B[n][k]A[i][1] * B[1][k] + \cdots + A[i][n] * B[n][k]$$

and print the result.

Your solution

```
n, m, r = [int(i) for i in input().split()]

a = [[int(j) for j in input().split()] for i in range(n)]

b = [[int(j) for j in input().split()] for i in range(m)]

c = [[0 for row in range(r)] for col in range(n)]

for i in range(n):

    for j in range(r):

        for k in range(m):

            c[i][j] += a[i][k]*b[k][j]

print('\n'.join([' '.join([str(i) for i in row]) for row in c]))
```