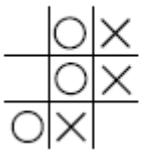
Матрицы

Что такое матрицы?

Многие программы работают с данными, организованными в виде таблиц. Например, при составлении программы для игры в крестики-нолики нужно запоминать состояние каждой клетки квадратной доски. Можно поступить так: пустым клеткам присвоить код «–1», клетке, где стоит нолик – код 0, а клетке с крестиком – код 1. Тогда информация о состоянии поля может быть записана в виде таблицы:



	0	1	2
0	-1	0	1
1	-1	0	1
2	0	1	-1

Такие таблицы называются *матрицами* или двухмерными массивами. Каждый элемент матрицы, в отличие от обычного (линейного) массива имеет два индекса – номер строки и номер столбца. На рисунке серым фоном выделен элемент, находящийся на пересечении второй строки и третьего столбца.

Матрица — это прямоугольная таблица, составленная из элементов одного типа (чисел, строк ит.д.). Каждый элемент матрицы имеет два индекса – номера строки и столбца. Поскольку в Python нет массивов, то нет и матриц в классическом понимании. Для того, чтобы работать с таблицами, используют списки. Двухмерная таблица хранится как список, каждый элемент которого тоже представляет собой список («список списков»). Например, таблицу, показанную на рисунке, можно записать так:

```
A = [[-1, 0, 1],
     [-1, 0, 1],
     [0, 1, -1]]
```

Этот оператор можно записать и в одну строчку:

```
A = [[-1, 0, 1], [-1, 0, 1], [0, 1, -1]]
```

но первый способ более нагляден, если мы задаём начальное значения для матрицы.

Простейший способ вывода матрицы – с помощью одного вызова функции **print**:

```
print ( A )
```

В этом случае матрица выводится в одну строку, что не очень удобно. Поскольку человек воспринимает матрицу как таблицу, лучше и на экран выводить её в виде таблицы. Для этого можно написать такую процедуру (в классическом стиле):

```
def printMatrix ( A ):
  for i in range ( len(A) ):
    for j in range ( len(A[i]) ):
      print ( "{:4d}".format(A[i][j]), end = "" )
    print ()
```

Здесь **i** – это номер строки, а **j** – номер столбца; **len(A)** – это число строк в матрице, а **len(A[i])** – число элементов в строке **i** (совпадает с числом столбцов, если матрица прямоугольная). Формат вывода «**4d**» означает «вывести целое число в 4-х позициях», послед выводаочередного числа не делаем перевод строки (**end=""**), а после вывода всей строки – делаем (**print()**).

Можно написать такую функцию и в стиле Python:

```
def printMatrix ( A ):
  for row in A:
    for x in row:
      print ( "{:4d}".format(x), end = "" )
    print ()
```

Здесь первый цикл перебирает все строки в матрице; каждая из них по очереди попадает в переменную **row**. Затем внутренний цикл перебирает все элементы этой строки и выводит их на экран.

Иногда нужно создать в памяти матрицу заданного размера, заполненную некоторыми начальными значениями, например, нулями[19]. Первая мысль – использовать такой алгоритм, использующий операцию повторения «*»:

```
N = 3
M = 2
# неверное создание матрицы!
row = [0]*M          # создаём список-строку длиной M
A = [row]*N          # создаём массив (список) из N строк
```

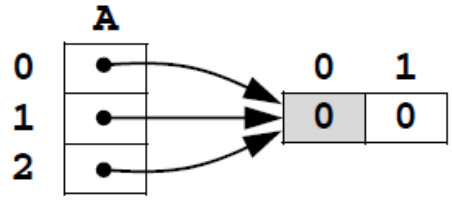
Однако этот способ работает неверно из-за особенностей языка Python. Например, если после этого выполнить присваивание

```
A[0][0] = 1
```

мы увидим, что *все* элементы столбца 0, то есть **A[0][0]**, **A[1][0]** и т.д. стали равны 1. Дело в том, что матрица – это список ссылок на списки-строки (список адресов строк). При выполнении оператора

```
A = [row]*N
```

транслятор создаёт в памяти одну единственную строку, на которую устанавливает все ссылки:



Естественно, что когда мы меняет элемент 0 в этой строке (он выделен серым фоном на рисунке), меняются и все элементы с номером 0 в каждой строке.

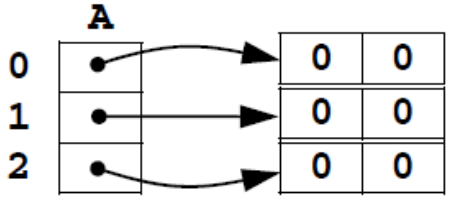
Для создания полноценной матрицы нам нужно как-то заставить транслятор создать все строки в памяти как разные объекты. Для этого сначала создадим пустой список, а потом будем в цикле добавлять к нему (с помощью метода **append**) новые строки, состоящие из нулей:

```
A = []
for i in range(N):
  A.append ( [0]*M )
```

или так, с помощью генератора:

```
A = [[0]*M for i in range(N)]
```

Теперь все строки расположены в разных местах памяти:



Каждому элементу матрицы можно присвоить любое значение. Поскольку индексов два, для заполнения матрицы нужно использовать вложенный цикл. Далее будем считать, что существует матрица **A**, состоящая из **N** строк и **M** столбцов, а **i** и **j** – целочисленные переменные, обозначающие индексы строки и столбца. В этом примере матрица заполняется случайными числами и выводится на экран:

```
import random
for i in range(N):
  for j in range(M):
    A[i][j] = random.randint ( 20, 80 )
    print ( "{:4d}".format(A[i][j]), end = "" )
  print()
```

Такой же двойной цикл нужно использовать для перебора всех элементов матрицы. Вот как вычисляется сумма всех элементов:

```
s = 0
for i in range(N):
  for j in range(M):
    s += A[i][j]
print ( s )
```

Поскольку мы не изменяем элементы матрицы, более красиво решать эту задачу в стиле Python:

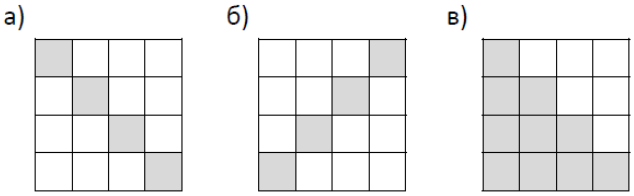
```
s = 0
for row in A:
  s += sum(row)
print ( s )
```

В цикле перебираются все строки матрицы A, каждая из них по очереди записывается в переменную **row**. В теле цикла сумма элементов строки прибавляется к значению переменной **s**.

Обработка элементов матрицы

Покажем, как можно обработать (например, сложить) некоторые элементы квадратной матрицы **A**, содержащей **N** строк и **N** столбцов.

На рис. *а* выделена главная диагональ матрицы, на рис. *б* – вторая (побочная) диагональ, на рис. *в* – главная диагональ и все элементы под ней.



Главная диагональ – это элементы **A[0,0]**, **A[1,1]**, …, **A[N-1,N-1]**, то есть номер строки равен номеру столбца. Для перебора этих элементов нужен один цикл:

```
for i in range(N):
  # работаем с A[i,i]
```

Элементы побочной диагонали – это **A[0,N-1]**, **A[1,N-2]**, …, **A[N-1,0]**. Заметим, что сумма номеров строки и столбца для каждого элемента равна **N-1**, поэтому получаем такой цикл перебора:

```
for i in range(N):
  # работаем с A[i,N-1-i]
```

В случае *в* (обработка всех элементов на главной диагонали и под ней) нужен вложенный цикл: номер строки будет меняться от 0 до **N-1**, а номер столбца для каждой строки **i** – от 0 до **i**:

```
for i in range(N):
  for j in range( i+1 ):
    # работаем с A[i,j]
```

Чтобы переставить столбцы матрицы, достаточно одного цикла. Например, переставим столбцы 2 и 4, используя вспомогательную переменную **c**:

```
for i in range(N):
  c = A[i][2]
  A[i][2] = A[i][4]
  A[i][4] = c
```

или, используя возможности Python:

```
for i in range(N):
  A[i][2], A[i][4] = A[i][4], A[i][2]
```

Переставить две строки можно вообще без цикла, учитывая, что **A[i]** – это ссылка на список элементов строки **i**. Поэтому достаточно просто переставить ссылки. Оператор

```
A[i], A[j] = A[j], A[i]
```

переставляет строки матрицы с номерами **i** и **j**.

Для того, чтобы создать копию строки с номером **i**, нельзя делать так (подумайте, почему?):

R = A[i]

а вместо этого нужно создать копию в памяти:

R = A[i][:]

Построить копию столбца с номером **j** несколько сложнее, так как матрица расположена в памяти по строкам. В этой задаче удобно использовать генератор:

```
C = [ row[j] for row in A ]
```

В цикле перебираются все строки матрицы **A**, они по очереди попадают в переменную **row**. Генератор выбирает из каждой строки элемент с номером **j** и составляет список из этих значений.

С помощью генератора легко выделить в отдельный массив элементы главной диагонали:

```
D = [ A[i][i] for i in range(N) ]
```

Здесь предполагается, что матрица **M** состоит из **N** строк и **N** столбцов.