Занятие 9. Двумерные массивы

1. Обработка и вывод вложенных списков

Часто в задачах приходится хранить прямоугольные таблицы с данными. Такие таблицы называются матрицами или двумерными массивами. В языке программирования Питон таблицу можно представить в виде списка строк, каждый элемент которого является в свою очередь списком, например, чисел. Например, приведём программу, в которой создаётся числовая таблица из двух строк и трех столбцов, с которой производятся различные действия.

запустить выполнить пошагово

```
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)

```

Здесь первая строка списка `a[0]` является списком из чисел `[1, 2, 3]`. То есть `a[0][0] == 1`, значение `a[0][1] == 2`, `a[0][2] == 3`, `a[1][0] == 4`, `a[1][1] == 5`, `a[1][2] == 6`.

Для обработки и вывода списка, как правило, используют два вложенных цикла. Первый цикл перебирает номер строки, второй цикл бежит по элементам внутри строки. Например, вывести двумерный числовой список на экран построчно, разделяя числа пробелами внутри одной строки, можно так:

запустить выполнить пошагово

```python
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()

```

Однажды мы уже пытались объяснить, что переменная цикла for в Питоне может перебирать не только диапазон, создаваемый с помощью функции `range()`, но и вообще перебирать любые элементы любой последовательности. Последовательностями в Питоне являются списки, строки, а также некоторые другие объекты, с которыми мы пока не встречались. Продемонстрируем, как выводить двумерный массив, используя это удобное свойство цикла `for`:

запустить выполнить пошагово

```
a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
for row in a:
    for elem in row:
        print(elem, end=' ')
    print()

```

Естественно, для вывода одной строки можно воспользоваться методом `join()`:

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

```

Используем два вложенных цикла для подсчета суммы всех чисел в списке:

запустить выполнить пошагово

```
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)

```

Или то же самое с циклом не по индексу, а по значениям строк:

запустить выполнить пошагово

```
a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
s = 0
for row in a:
    for elem in row:
        s += elem
print(s)

```

2. Создание вложенных списков

Пусть даны два числа: количество строк `n` и количество столбцов `m`. Необходимо создать список размером `n` × `m`, заполненный нулями.

Очевидное решение оказывается неверным:

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

```

В этом легко убедиться, если присвоить элементу `a[0][0]` значение `5`, а потом вывести значение другого элемента `a[1][0]` — оно тоже будет равно 5. Дело в том, что `[0] * m` возвращает ссылку на список из `m` нулей. Но последующее повторение этого элемента создает список из `n` элементов, которые являются ссылкой на один и тот же список (точно так же, как выполнение операции `b = a` для списков не создает новый список), поэтому все строки результирующего списка на самом деле являются одной и той же строкой.

В визуализаторе обратите внимание на номер id у списков. Если у двух списков id совпадает, то это на самом деле один и тот же список в памяти.

запустить выполнить пошагово

```
n = 3
m = 4
a = [[0] * m] * n
a[0][0] = 5
print(a[1][0])

```

Таким образом, двумерный список нельзя создавать при помощи операции повторения одной строки. Что же делать?

Первый способ: сначала создадим список из `n` элементов (для начала просто из `n` нулей). Затем сделаем каждый элемент списка ссылкой на другой одномерный список из `m` элементов:

запустить выполнить пошагово

```
n = 3
m = 4
a = [0] * n
for i in range(n):
    a[i] = [0] * m

```

Другой (но похожий) способ: создать пустой список, потом `n` раз добавить в него новый элемент, являющийся списком-строкой:

запустить выполнить пошагово

```
n = 3
m = 4
a = []
for i in range(n):
    a.append([0] * m)

```

Но еще проще воспользоваться генератором: создать список из `n` элементов, каждый из которых будет списком, состоящих из `m` нулей:

запустить выполнить пошагово

```
n = 3
m = 4
a = [[0] * m for i in range(n)]

```

В этом случае каждый элемент создается независимо от остальных (заново конструируется список `[0] * m` для заполнения очередного элемента списка), а не копируются ссылки на один и тот же список.

3. Ввод двумерного массива

Пусть программа получает на вход двумерный массив в виде `n` строк, каждая из которых содержит `m` чисел, разделенных пробелами. Как их считать? Например, так:

запустить выполнить пошагово

```
# в первой строке ввода идёт количество строк массива
n = int(input())
a = []
for i in range(n):
    a.append([int(j) for j in input().split()])

```

Или, без использования сложных вложенных вызовов функций:

запустить выполнить пошагово

```python
# в первой строке ввода идёт количество строк массива
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)

```

Можно сделать то же самое и при помощи генератора:

запустить выполнить пошагово

```
# в первой строке ввода идёт количество строк массива
n = int(input())
a = [[int(j) for j in input().split()] for i in range(n)]

```

4. Пример обработки двумерного массива

Пусть дан квадратный массив из `n` строк и `n` столбцов. Необходимо элементам, находящимся на главной диагонали, проходящей из левого верхнего угла в правый нижний (то есть тем элементам `a[i][j]`, для которых `i==j`) присвоить значение `1`, элементам, находящимся выше главной диагонали – значение 0, элементам, находящимся ниже главной диагонали – значение 2. То есть необходимо получить такой массив (пример для `n==4`):

```
1 0 0 0
2 1 0 0
2 2 1 0
2 2 2 1

```

Рассмотрим несколько способов решения этой задачи. Элементы, которые лежат выше главной диагонали – это элементы `a[i][j]`, для которых `i<j`, а для элементов ниже главной диагонали `i>j`. Таким образом, мы можем сравнивать значения `i` и `j` и по ним определять значение `A[i][j]`. Получаем следующий алгоритм:

запустить выполнить пошагово

```python
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]))

```

Данный алгоритм плох, поскольку выполняет одну или две инструкции `if` для обработки каждого элемента. Если мы усложним алгоритм, то мы сможем обойтись вообще без условных инструкций.

Сначала заполним главную диагональ, для чего нам понадобится один цикл:

```
for i in range(n):
    a[i][i] = 1

```

Затем заполним значением `0` все элементы выше главной диагонали, для чего нам понадобится в каждой из строк с номером `i` присвоить значение элементам `a[i][j]` для `j = i+1 , ..., n-1` . Здесь нам понадобятся вложенные циклы:

```
for i in range(n):
    for j in range(i + 1, n):
        a[i][j] = 0

```

Аналогично присваиваем значение `2` элементам `a[i][j]` для `j = 0 , ..., i-1` :

```
for i in range(n):
    for j in range(0, i):
        a[i][j] = 2

```

Можно также внешние циклы объединить в один и получить еще одно, более компактное решение:

запустить выполнить пошагово

```
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]))

```

А вот такое решение использует операцию повторения списков для

построения очередной строки списка. `i` -я строка списка состоит из `i` чисел `2`, затем идет одно число `1`, затем идет `n-i-1` число `0`:

запустить выполнить пошагово

```
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]))

```

А можно заменить цикл на генератор:

запустить выполнить пошагово

```
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. Вложенные генераторы двумерных массивов

Для создания двумерных массивов можно использовать вложенные генераторы, разместив генератор списка, являющегося строкой, внутри генератора всех строк. Напомним, что сделать список из `n` строк и `m` столбцов можно при помощи генератора, создающего список из `n` элементов, каждый элемент которого является списком из `m` нулей:

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

```

Но при этом внутренний список также можно создать при помощи, например, такого генератора: `[0 for j in range(m)]`. Вложив один генератор в другой, получим вложенные генераторы:

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

```

Но если число 0 заменить на некоторое выражение, зависящее от `i` (номер строки) и `j` (номер столбца), то можно получить список, заполненный по

некоторой формуле.

Например, пусть нужно задать следующий массив (для удобства добавлены дополнительные пробелы между элементами):

```
 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

```

В этом массиве `n = 5` строк, `m = 6` столбцов, и элемент в строке `i` и столбце `j` вычисляется по формуле: `a[i][j] = i * j` .

Для создания такого массива можно использовать генератор:

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

```

Задачи на informatics.mccme.ru