# Урок 06.03. Вложенные циклы

В программирование очень часто приходится использовать вложенность, и давайте рассмотрим несколько задач на эту тему.

## Задача «Матрица»

Необходимо вывести таблицу размером n на n, где все элементы, стоящие на главной диагонали (главная диагональ – воображаемая линия из левого верхнего угла в правый нижний угол, вторая диагональ – из правого верхнего угла в левый нижний), равны 1, все элементы выше главной диагонали равны 0, а ниже – 2.

И как всегда мы будем решать задачу поэтапно. Разобьём нашу программу на следующие этапы:

– ввод данных и вывод таблицы из нулей

– изменение программы для вывода 1 (будем искать формулу для диагонали);

– изменение программы для вывода 2.

Давайте для начала выведем таблицу n на n из 0:

n = int(input(‘Введите размер таблицы: ’))

for row in range(n):

for col in range(n):

print(‘0’, end=' ')

print()

Отлично. Теперь переходим ко второй задаче. Как же определить диагональ? Если внимательно посмотреть на нашу таблицу, то можно увидеть, что диагональные элементы это такие элементы, у которых номер строчки и номер колонки совпадает (выделено желтым цветом). При этом введем следующее обозначение (строка, столбец) или по-английски (row, column):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| (0, 0) | (0, 1) | (0, 2) | … | (0, n-1) | (0, n) |
| (1, 0) | (1, 1) | (1, 2) | … | (0, n-1) | (0, n) |
| (2, 0) | (2, 1) | (2, 2) | … | (0, n-1) | (0, n) |
| … | … | … | … | … | … |
| (n-1, 0) | (n-1, 1) | (n-1, 2) | … | (n-1, n-1) | (n-1, n) |
| (n, 0) | (n, 1) | (n, 2) | … | (n, n-1) | (n, n) |

Тогда получается, что если номер строки совпадает с номером колонки, то нужно выводить 1, иначе выводить 0. А это получается уже условие на вывод. Так и запишем

n = int(input('Введите размер таблицы: '))

for row in range(n):

for col in range(n):

if row == col:

print('1', end=' ')

else:

print('0', end=' ')

print()

Так, теперь у нас есть таблица (она же матрица) с единичками по диагонали. Идём дальше.

Теперь нам нужно ниже главной диагонали поставить двойки. И, как вы могли уже догадаться, то нам нужно как-то поменять место в программе, которое отвечает за вывод на экран, а значит опять написать новое условие. А теперь давайте подумаем, какому условию отвечают элементы в таблице, стоящие ниже главной диагонали (выделено зеленым цветом)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| (0, 0) | (0, 1) | (0, 2) | … | (0, n-1) | (0, n) |
| (1, 0) | (1, 1) | (1, 2) | … | (0, n-1) | (0, n) |
| (2, 0) | (2, 1) | (2, 2) | … | (0, n-1) | (0, n) |
| … | … | … | … | … | … |
| (n-1, 0) | (n-1, 1) | (n-1, 2) | … | (n-1, n-1) | (n-1, n) |
| (n, 0) | (n, 1) | (n, 2) | … | (n, n-1) | (n, n) |

Все элементы, которые находятся ниже главной диагонали отвечают условию, что номер колонки всегда меньше номера строчки, тогда наша программа преобразится:

n = int(input('Введите размер таблицы: '))

for row in range(n):

for col in range(n):

if row == col:

print('1', end=' ')

elif row > col:

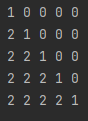
print('2', end=' ')

else:

print('0', end=' ')

print()

Вот так у нас и получилась наша программа.



## Задача «Дорога»

Допустим, мы делаем игру про гонки и нам необходимо написать программу, которая «нарисует» дорогу

/ | \

/ | \

/ | \

/ | \

--------------------------

/ | \

/ | \

/ | \

/ | \

Опять разделим нашу задачу на этапы:

– начнем с рисования вертикальной и горизонтальных полос;

– потом нарисуем правую обочину;

– и в конце нарисуем левую обочину.

Приступим. Начнем с того что определим размер нашей дороги (количество строчек и столбцов). Хотя нам и дан рисунок дороги, но нигде не сказано, что он не может меняться, и потом наша дорога должна уметь изменять размеры, тогда нам нужно будет сделать программу, в которой количество строчек и столбцов задается в начале программы. Нарисуем горизонтальную и вертикальную линию посередине. И напишем нашу программу

x\_lim = 50

y\_lim = 20

delta = 2

for row in range(y\_lim):

for col in range(x\_lim):

if col == x\_lim // 2 and row == y\_lim // 2:

print('+', end='')

elif col == x\_lim // 2:

print('|', end='')

elif row == y\_lim // 2:

print('—', end='')

else:

print(' ', end='')

print()

Первый этап сделали, приступим ко-второму. Тут самое сложное – нарисовать диагонали. Обычную диагональ мы знаем, как нарисовать – она рисуется если колонка совпадает со строчкой. Напишем такую программу

x\_lim = 50

y\_lim = 20

delta = 2

for row in range(y\_lim):

for col in range(x\_lim):

if col == x\_lim // 2 and row == y\_lim // 2:

print('+', end='')

elif col == x\_lim // 2:

print('|', end='')

elif row == y\_lim // 2:

print('—', end='')

elif row == col:

print('\\', end='')

else:

print(' ', end='')

print()

Мы получили диагональ, но немного не там. Как же её подвинуть? На самом деле очень просто, нужно в условии (elif row == col:) к колонке прибавить (или отнять) нужное число. А нужное число это половина ширины (x\_lim // 2) и добавка в виде отступа от центра (delta), тогда наша программу будет выглядеть следующим образом

x\_lim = 50

y\_lim = 20

delta = 2

for row in range(y\_lim):

for col in range(x\_lim):

if col == x\_lim // 2 and row == y\_lim // 2:

print('+', end='')

elif col == x\_lim // 2:

print('|', end='')

elif row == y\_lim // 2:

print('—', end='')

elif row == col - (x\_lim // 2 + delta):

print('\\', end='')

else:

print(' ', end='')

print()

Альтернативным решением является следующее. Нам нужно рисовать символ правой обочины «\» тогда, когда номер колонки равен половине ширины плюс дельта и плюс смещение от номера строки: col == x\_lim // 2 + (delta + row). На самом деле это одно и тоже математическое выражения, однако к нему привели разные подходы в рассуждении.

Используя третье рассуждение попробуйте сами вывести формулу. Если всё же долго не получается, то ниже представлена программа.

x\_lim = 25

y\_lim = 9

delta = 2

for row in range(y\_lim):

for col in range(x\_lim):

if col == x\_lim // 2 and row == y\_lim // 2:

print('+', end='')

elif col == x\_lim // 2:

print('|', end='')

elif row == y\_lim // 2:

print('—', end='')

elif col == x\_lim // 2 + (delta + row):

print('\\', end='')

elif col == x\_lim // 2 - (delta + row):

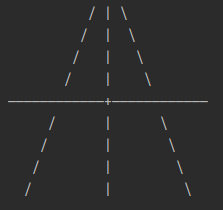
print('/', end='')

else:

print(' ', end='')

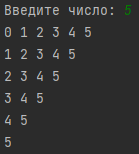
print()

Всё. Наша дорога готова.



## Задача «Лестница чисел»

Пользователь вводит число N. Необходимо написать программу, которая по этому числу выводит вот такую лестницу из чисел:



Опять разобьем всё на этапы:

– ввод данных, вывод пяти строчек чисел от 0 до N;

– изменение программы так, чтобы выводились только нужные числа.

Решение первого этапа, думаю, не вызывает проблем:

n = int(input('Введите число: '))

for row in range(n + 1):

for col in range(0, n + 1):

print(col, end=' ')

print()

Обращаю ваше внимание на то, что в range окончание пишем n + 1.

Приступи ко второму этапу. Возможно вы хотите сделать это через условие, вот так:

n = int(input('Введите число: '))

for row in range(n + 1):

for col in range(0, n + 1):

if col >= row:

print(col, end=' ')

print()

И эта программа будет работать. Но у неё есть существенный недостаток. Она «впустую» считает счетчик во втором цикле. Пока нам «пустая» работа программы не мешает и не тормозит, но когда мы будем писать более сложный код, то это может потом оказаться очень медленным кодом. Так как же нам избавиться от пустых вычислений счетчика внутреннего цикла? Если немного подумать, то мы увидим, что цифры выводятся на экран (тогда счетчик и считается «впустую») каждый раз с числа равного номеру строки. Таким образом, у нас получается, что внутренний цикл должен начинаться не с 0, а с номера строки, тогда и пропадает необходимость в условии. И вот эта программа

n = int(input('Введите число: '))

for row in range(n + 1):

for col in range(row, n + 1):

print(col, end=' ')

print()

И вот наша программа опять готова.

## Задача «Банкомат»

Допустим нам нужно написать программу для банкомата. Человек подходит, вставляет свою банковскую карту. У пользователя есть три попытки на ввод PIN‑код. А мы знаем, что после трех попыток ввода неправильного PIN-кода карточка блокируется. Будем решать задачу через цикл for.

for attempt in range(1, 4):

pincode = int(input('Введите pin-код: '))

if pincode == 1234:

print('pin-код верный. Держите вашу зарплату!')

break

print('Не верный pin-код. Осталось попыток:', 3 - attempt)

Теперь нам нужно дописать программу, чтобы она выводила сообщение о блокировке карточки в случае неверного ввода. Если решать классическим путем, то нам понадобится переменная, которая будет хранить состояние введен правильный пароль или нет. Напишем такую программу:

flag\_valid = False

for attempt in range(1, 4):

pincode = int(input('Введите pin-код: '))

if pincode == 1234:

print('pin-код верный. Держите вашу зарплату!')

flag\_valid = True

break

print('Не верный pin-код. Осталось попыток:', 3 - attempt)

if not flag\_valid:

print('Ваша карта заблокирована!')

Но как вы уже привыкли в python есть множество удобных возможностей, и эта ситуация не исключение. У циклов, как и у условного оператора есть условие else. И тогда можно убрать нашу новую переменную.

for attempt in range(1, 4):

pincode = int(input('Введите pin-код: '))

if pincode == 1234:

print('pin-код верный. Держите вашу зарплату!')

break

print('Не верный pin-код. Осталось попыток:', 3 - attempt)

else:

print('Ваша карта заблокирована!')

Может показаться немного странным такое использование оператора else, однако если разобраться, то получается, что блок else выполняется тогда, когда цикл полностью проделал свою работу, или больше не может выполняться, т.е. сработало условие на окончание цикла.