Вложенные циклы

Рассмотрим задачи, в которых в качестве оператора цикла используется другой цикл.

Пример 1. Для заданного натурального числа n требуется найти все тройки натуральных чисел a, b, c таких, что $a + b + c = n$. Решение:

```
for a in range(1, n - 1):
    for b in range(1, n - 1):
        c = n - a - b
        if c > 0:
            print(a,'+',b,'+',c,'=',n)
```

Данный пример одновременно показывает, как можно использовать вложенные циклы и сокращать их количество. Подумайте, почему параметры обоих циклов изменяются именно до $n - 2$ включительно? Давайте рассмотрим порядок выполнения этих циклов. Пусть $n = 5$. Тогда переменная a сначала принимает значение 1, а переменная b пробегает значения от 1 до 3. Запишем в таблице результат работы программы на каждом из таких шагов:

a	*b*	*c*	результат
1	1	3	1 + 1 + 3 = 5
1	2	2	1 + 2 + 3 = 5
1	3	1	1 + 3 + 1 = 5

Затем переменная a принимает значения 2 и 3, а b заново пробегает значения от 1 до 3:

a	*b*	*c*	результат
2	1	2	2 + 1 + 2 = 5
2	2	1	2 + 2 + 1 = 5
2	3	0	
3	1	1	3 + 1 + 1 = 5
3	2	0	
3	3	-1	

В случаях, когда c окажется меньше или раной 0 ничего напечатано не будет.

Эту программу можно еще упростить, изменив параметр второго из циклов

```
for a in range(1, n - 1):
    for b in range(1, n - a):
        c = n - a - b
        print(a, '+', b, '+', c, '=', n)
```

В этом случае c всегда будет больше или равно 0:
так как $b < n - a$, то $a + b < n$ и $n - a - b > 0$.

Пример 2. Требуется распечатать все трехзначные числа, в которых вторая цифра больше первой, а третья больше второй. Решение:

```
for i in range(1, 8):
    for j in range(i + 1, 9):
        for k in range(j + 1, 10):
            print(i, j, k, sep = '')
```

В данном решении также удалось избежать использования условного оператора, а также операций деления для выделения цифр из числа. Этот пример показывает, что в задачах по информатике, для того чтобы получить ответ, не всегда нужно оперировать именно теми объектами, о которых идет речь в условии задачи. В этих задачах совершенно излишним было бы перебирать все трехзначные числа, выделять из них цифры и сравнивать их между собой. Более того, формировать из подходящих цифр число, только для того чтобы его распечатать, тоже не нужно: печать нескольких цифр подряд без разделителей приведет к тому, что зрительно эти цифры как раз и образуют нужное число.

Пример 3. Разложение натурального числа $n > 1$ на простые сомножители.

Задача относится к числу задач базовых проблем, которые можно рекомендовать для обязательного рассмотрения. Алгоритм решения задачи следующий. Возьмем первое простое число — 2. Пока *n* делится на 2, то будем делить его на 2 и двойки сразу печатать. Если число на 2 уже не делится, то будем пытаться делить его на 3 и т.д. Интересно, что проверять при этом делители на простоту не нужно! Дело в том, что если число уже разделили на 2 и на 3 максимально возможное число раз, то ни на 4, ни на 6, ни на 9 оно не разделится автоматически. Поэтому делители можно проверять подряд, но делать это только до квадратного корня из исходного числа *n* (при программировании мы постараемся избежать непосредственного извлечения корня). Если оставшееся число больше единицы, то оно тоже простое и его нужно напечатать.

Приведем фрагмент соответствующей программы:

```
d = 2
while d * d <= n:
    while n % d == 0:
        print(d)
        n //= d
    d += 1
if n > 1:
    print(n)
```

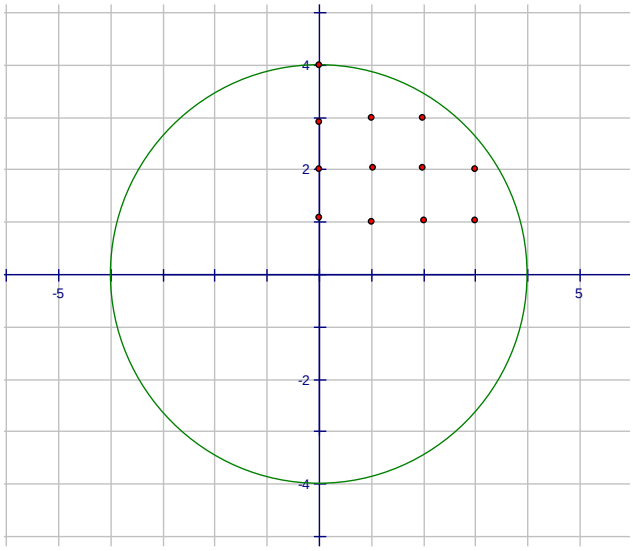
Пример 4. Пусть требуется подсчитать число точек с целочисленными координатами, находящихся внутри и на границе круга с центром в начале координат и заданным радиусом *r*.

На вход программе подается целое неотрицательное число $r \leq 1\,000\,000$. Выведите количество искомых точек в этом круге.

Пример входных данных	**Пример выходных данных**
2	13

Перебрать все точки в данном случае не удастся. Решим задачу для четверти круга. Целочисленные точки будем суммировать по столбцам. На оси координат *Oy* выше оси *Ox* лежит ровно *r* целочисленных точек круга. В каждом следующем столбце будет либо

столько же целочисленных точек круга, либо меньше (см. *рис.*). На сколько именно меньше — проще всего выяснить, вычитая точки по одной и подставляя результат в уравнение окружности (на самом деле, в неравенство, которое позволяет определить, лежит ли точка внутри круга). В этом случае все вычисления будут производится точно. Количество точек в столбце можно определять и по формуле, но тогда могут возникнуть проблемы с точностью. Кроме того, несложно доказать, что общее количество таких проверок окажется порядка r, потому что количество точек в столбце будет только уменьшаться и принимать значения от r до 0.



Приведем возможный вариант соответствующей программы:

```
r = in(input())
ans = r
y = r
for x in range(r):
    while r*r < x*x + y*y:
        s -= 1
    ans += y
ans = 4 * ans + 1
# начало координат учитывается отдельно
print(ans)
```