Функции

Ранее была задача вычисления числа сочетаний из n элементов по k, для чего необходимо вычисление факториалов трех величин: n, k и n-k. Для этого можно сделать три цикла, что приводит к увеличению размера программы за счет трехкратного повторения похожего кода. Вместо этого лучше сделать одну *функцию*, вычисляющую факториал любого данного числа n и трижды использовать эту функцию в своей программе. Соответствующая функция может выглядеть так:

```
def factorial(n):
    f = 1
    for i in range(2, n + 1):
        f *= i
    return f
```

Этот текст должен идти в начале программы, вернее, до того места, где мы захотим воспользоваться функцией `factorial`. Первая строчка этого примера является описанием нашей функции. `factorial` — идентификатор, то есть имя нашей функции. После идентификатора в круглых скобках идет список параметров, которые получает наша функция. Список состоит из перечисленных через запятую идентификаторов параметров. В нашем случае список состоит из одной величины `n`. В конце строки ставится двоеточие.

Далее идет тело функции, оформленное в виде блока, то есть с отступом. Внутри функции вычисляется значение факториала числа `n` и оно сохраняется в переменной `f`. Функция завершается инструкцией `return f`, которая завершает работу функции и возвращает значение переменной `f`. Инструкция `return` может встречаться в произвольном месте функции, ее исполнение завершает работу функции и возвращает указанное значение в место вызова. Если функция не возвращает значения, то инструкция `return` используется без возвращаемого значения, также в функциях, не возвращающих значения, инструкция `return` может отсутствовать.

Теперь мы можем использовать нашу функцию несколько раз. В этом примере мы трижды вызываем функцию `factorial` для вычисления трех факториалов: `factorial(n)`, `factorial(k)`, `factorial(n-k)`.

```
n = int(input())
k = int(input())
print(factorial(n) // (factorial(k) * factorial(n - k)))
```

Мы также можем, например, объявить функцию `binomial`, которая принимает два целочисленных параметра `n` и `k` и вычисляет число сочетаний из `n` по `k`:

```
def binomial(n, k)
    return factorial(n) // (factorial(k) * factorial(n - k))
```

Тогда в нашей основной программе мы можем вызвать функцию `binomial` для нахождения числа сочетаний:

```
print(binomial(n, k))
```

Вернемся к задаче нахождения наибольшего из двух или трех чисел. Функцию нахождения максимума из двух чисел можно написать так:

```
def max(a, b):
    if a > b:
        return a
    else:
        return b
```

Теперь мы можем реализовать функцию `max3`, находящую максимум трех чисел:

```
def max3(a, b, c):
    return max(max(a, b), c)
```

Функция `max3` дважды вызывает функцию `max` для двух чисел: сначала, чтобы найти максимум из `a` и `b`, потом чтобы найти максимум из этой величины и `c`.

Локальные и глобальные переменные

Внутри функции можно использовать переменные, объявленные вне этой функции

```
def f():
    print(a)
a = 1
f()
```

Здесь переменной `a` присваивается значение 1, и функция `f` печатает это значение, несмотря на то, что выше функции `f` эта переменная не инициализируется. Но в момент вызова функции `f` переменной `a` уже присвоено значение, поэтому функция `f` может вывести его на экран.

Такие переменные (объявленные вне функции, но доступные внутри функции) называются *глобальными*.

Но если инициализировать какую-то переменную внутри функции, использовать эту переменную вне функции не удастся. Например:

```
def f():
    a = 1
f()
print(a)
```

Получим `NameError: name 'a' is not defined`. Такие переменные, объявленные внутри функции, называются *локальными*. Эти переменные становятся недоступными после выхода из функции.

Интересным получится результат, если попробовать изменить значение глобальной переменной внутри функции:

```
def f():
    a = 1
    print(a)
a = 0
f()
print(a)
```

Будут выведены числа 1 и 0. То есть несмотря на то, что значение переменной `a` изменилось внутри функции, то вне функции оно осталось прежним! Это сделано в целях “защиты” глобальных переменных от случайного изменения из функции (например, если функция будет вызвана из цикла по переменной `i`, а в этой функции будет использована переменная `i` также для организации цикла, то эти переменные должны быть различными). То есть если внутри функции модифицируется значение некоторой переменной, то переменная с таким именем становится локальной переменной, и ее модификация не приведет к изменению глобальной переменной с таким же именем.

Более формально: интерпретатор Питон считает переменную локальной, если внутри нее есть хотя бы одна инструкция, модифицирующая значение переменной (это может быть оператор `=`, `+=` и т.д., или использование этой переменной в качестве параметра цикла `for`, то эта переменная считается локальной и не может быть использована до инициализации. При этом даже если инструкция, модицифицирующая переменную никогда не будет выполнена: интерпретатор это проверить не может, и переменная все равно считается локальной. Пример:

```
def f():
    print(a)
    if False:
        a = 0
a = 1
f()
```

Возникает ошибка: `UnboundLocalError: local variable 'a' referenced before assignment`. А именно, в функции `f` идентификатор `a` становится локальной переменной, т.к. в функции есть команда, модифицирующая переменную `a`, пусть даже никогда и не выполняющийся (но интерпретатор не может это отследить). Поэтому вывод переменной `a` приводит к обращению к неинициализированной локальной переменной.

Чтобы функция могла изменить значение глобальной переменной, необходимо объявить эту переменную внутри функции, как глобальную, при помощи ключевого слова `global`:

```
def f():
    global a
    a = 1
    print(a)
a = 0
f()
print(a)
```

В этом примере на экран будет выведено 1 1, так как переменная `a` объявлена, как глобальная, и ее изменение внутри функции приводит к тому, что и вне функции переменная будет доступна.

Тем не менее, лучше не изменять значения глобальных переменных внутри функции. Если функция должна поменять какую-то переменную, то как правило это лучше сделать, как значение, возвращаемое функцией.

Если нужно, чтобы функция вернула не одно значение, а два или более, то для этого функция может вернуть кортеж из двух или нескольких значений:

```
return (a, b)
```

Тогда результат вызова функции тоже нужно присваивать кортежу:

```
(n, m) = f(a, b)
```

Упражнения

A: Минимум 4 чисел

Напишите функцию `min4(a, b, c, d)`, вычисляющую минимум четырех чисел, которая не содержит инструкции `if`, а использует **стандартную** функцию `min` от двух аргументов. Считайте четыре целых числа и выведите их минимум.

Ввод	Вывод
5 1 7 3	1

B: Длина отрезка

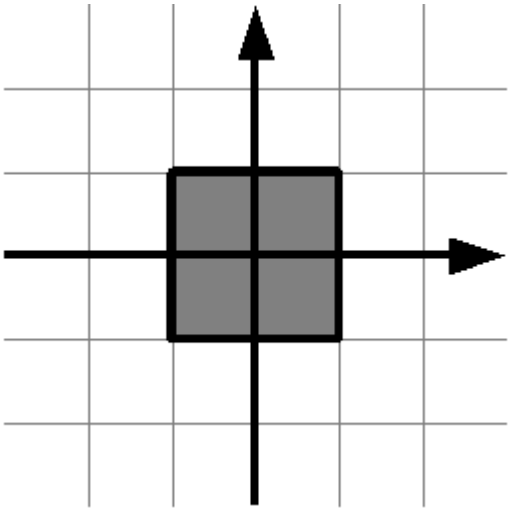
Даны четыре действительных числа: x_1, y_1, x_2, y_2. Напишите функцию `distance(x1, y1, x2, y2)`, вычисляющую расстояние между точкой (x_1, y_1) и (x_2, y_2). Считайте четыре действительных числа и выведите результат работы этой функции.

Ввод	Вывод
0 0	1.41421

1 1	

C: Принадлежит ли точка квадрату - 1

Даны два действительных числа x и y. Проверьте, принадлежит ли точка с координатами (x, y) заштрихованному квадрату (включая его границу). Если точка принадлежит квадрату, выведите слово `YES`, иначе выведите слово `NO`. На рисунке сетка проведена с шагом 1.



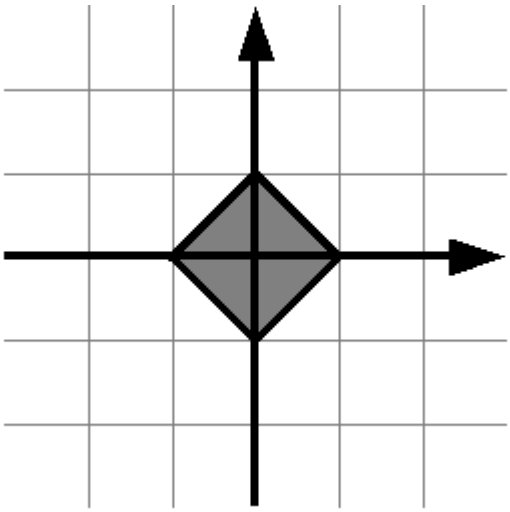
Решение должно содержать функцию `IsPointInSquare(x, y)`, возвращающую `True`, если точка принадлежит квадрату и `False`, если не принадлежит. Основная программа должна считать координаты точки, вызвать функцию `IsPointInSquare` и в зависимости от возвращенного значения вывести на экран необходимое сообщение.

Функция `IsPointInSquare` не должна содержать инструкцию `if`.

Ввод	Вывод
0 0	YES
3 -7	NO

D: Принадлежит ли точка квадрату - 2

Решите аналогичную задачу для такого квадрата:



Решение должно соответствовать требованиям для решения задачи C.

Ввод	Вывод
0 0	YES
1 1	NO

E: Принадлежит ли точка кругу

Даны пять действительных чисел: x, y, x_c, y_c, r. Проверьте, принадлежит ли точка (x, y) кругу с центром (x_c, y_c) и радиусом r.

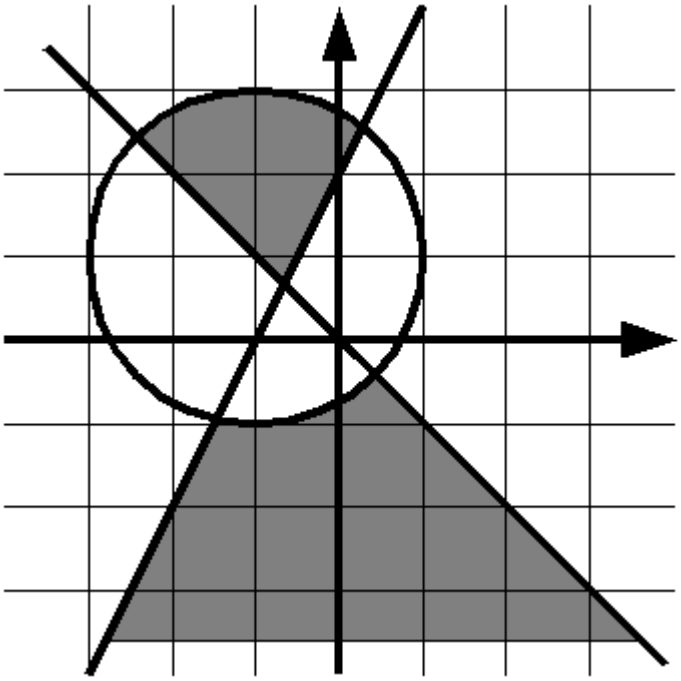
Решение оформите в виде функции `IsPointInCircle(x, y, xc, yc, r)`.

Решение должно соответствовать требованиям для решения задачи C.

Ввод	Вывод
0.5 0.5 0 0 1	YES
0.5 0.5 1 1 0.1	NO

F: Принадлежит ли точка области

Проверьте, принадлежит ли точка данной закрашенной области:



Решение оформите в виде функции `IsPointInArea(x, y)`.

Решение должно соответствовать требованиям для решения задачи C.

Ввод	Вывод
-1 2	YES
0 0	NO

G: Отрицательная степень

Дано действительное положительное число a и **целое** число n.

Вычислите a^n. Решение оформите в виде функции `power(a, n)`.

Стандартной функцией возведения в степень пользоваться нельзя.

Ввод	Вывод
2 3	8
2 -3	0.125

H: Сократите дробь

Даны два натуральных числа n и m. Сократите дробь $\frac{n}{m}$, то есть выведите два других числа p и q таких, что $\frac{n}{m} = \frac{p}{q}$ и дробь $\frac{p}{q}$ — несократимая.

Решение оформите в виде функции `ReduceFraction(n, m)`, получающей значения `n` и `m` и возвращающей кортеж из двух чисел.

Ввод	Вывод
12 16	3 4

I: Минимальный делитель числа

Дано натуральное число $n > 1$. Выведите его наименьший делитель, отличный от 1.

Решение оформите в виде функции `MinDivisor(n)`. Алгоритм должен иметь сложность $O(\sqrt{n})$.

Указание. Если у числа n нет делителя не превосходящего $\sqrt{n}$, то число n — простое и ответом будет само число n.

Ввод	Вывод
4	2
5	5

J: Проверка числа на простоту

Дано натуральное число $n > 1$. Проверьте, является ли оно простым. Программа должна вывести слово `YES`, если число простое и `NO`, если число составное.

Решение оформите в виде функции `IsPrime(n)`, которая возвращает `True` для простых чисел и `False` для составных чисел. Решение должно иметь сложность $O(\sqrt{n})$.

Ввод	Вывод
2	YES
4	NO

Рекурсия

Эпиграф:

```
def ShortStory():
    print("У попа была собака, он ее любил.")
    print("Она съела кусок мяса, он ее убил,")
    print("В землю закопал и надпись написал:")
    ShortStory()
```

Как мы видели выше, функция может вызывать другую функцию. Но функция также может вызывать и саму себя! Рассмотрим это на примере функции вычисления факториала. Хорошо известно, что $0! = 1$, $1! = 1$. А как вычислить величину $n!$ для большого n? Если бы мы могли вычислить величину $(n-1)!$, то тогда мы легко вычислим $n!$, поскольку $n! = n(n-1)!$. Но как вычислить $(n-1)!$? Если бы мы вычислили $(n-2)!$, то мы сможем вычисли и $(n-1)! = (n-1)(n-2)!$. А как вычислить $(n-2)!$? Если бы... В конце концов, мы дойдем до величины $0!$, которая равна 1. Таким образом, для

вычисления факториала мы можем использовать значение факториала для меньшего числа. Это можно сделать и в программе на Питоне:

```
def factorial (n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Подобный прием (вызов функцией самой себя) называется рекурсией, а сама функция называется рекурсивной.

Рекурсивные функции являются мощным механизмом в программировании. К сожалению, они не всегда эффективны (об этом речь пойдет позже). Также часто использование рекурсии приводит к ошибкам, наиболее распространенная из таких ошибок – бесконечная рекурсия, когда цепочка вызовов функций никогда не завершается и продолжается, пока не кончится свободная память в компьютере. Пример бесконечной рекурсии приведен в эпиграфе к этому разделу. Две наиболее распространенные причины для бесконечной рекурсии:

1. Неправильное оформление выхода из рекурсии. Например, если мы в программе вычисления факториала забудем поставить проверку `if n == 0`, то `factorial(0)` вызовет `factorial(-1)`, тот вызовет `factorial(-2)` и т.д.
2. Рекурсивный вызов с неправильными параметрами. Например, если функция `factorial(n)` будет вызывать `factorial(n)`, то также получиться бесконечная цепочка.

Поэтому при разработке рекурсивной функции необходимо прежде всего оформлять условия завершения рекурсии и думать, почему рекурсия когда-либо завершит работу.

Упражнения на рекурсию

K: Возведение в степень

Дано действительное положительное число a и целое неотрицательное число n. Вычислите a^n не используя циклы и стандартную функцию `pow`, а используя рекуррентное соотношение $a^n = a \cdot a^{n-1}$.

Решение оформите в виде функции `power(a, n)`.

Ввод	Вывод
2 3	8

L: Сложение без сложения

Напишите рекурсивную функцию `sum(a, b)`, возвращающую сумму двух целых неотрицательных чисел. Из всех арифметических операций допускаются только `+ 1` и `- 1`. Также нельзя использовать циклы.

Ввод	Вывод
2 2	4

M: Числа Фибоначчи

Напишите функцию `fib(n)`, которая по данному целому неотрицательному n возвращает n-е число Фибоначчи.

Ввод	Вывод
6	8

M+: Полезна ли рекурсия?

Запустите программы, вычисляющие числа Фибоначчи при помощи рекурсивного и нерекурсивного алгоритма. Сравните время их работы для n=10, 15, 20, 25, 30, 35. Объясните результат.

N: Число сочетаний

По данным числам n и k ($0 \le k \le n$) вычислите C_n^k. Для решения используйте рекуррентное соотношение $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$.

Решение оформите в виде функции `C(n, k)`.

Ввод	Вывод
6 3	20

O: Сумма последовательности

Дана последовательность чисел, завершающаяся числом 0. Найдите сумму всех этих чисел, не используя цикл.

Ввод	Вывод
1 7 9 0	17

P: Разворот последовательности

Дана последовательность целых чисел, заканчивающаяся числом 0. Выведите эту последовательность в обратном порядке.

При решении этой задачи нельзя пользоваться массивами и прочими динамическими структурами данных. Рекурсия вам поможет.

Ввод	Вывод
1 2 3 0	0 3 2 1

Q: Быстрое возведение в степень

Возводить в степень можно гораздо быстрее, чем за n умножений! Для этого нужно воспользоваться следующими рекуррентными соотношениями:

$a^n = (a^2)^{n/2}$ при четном n,

$a^n = a \cdot a^{n-1}$ при нечетном n.

Реализуйте алгоритм быстрого возведения в степень. Если вы все сделаете правильно, то сложность вашего алгоритма будет $O(\log n)$.

Ввод	Вывод
1.000000001 1000000000	2.71828

R: Алгоритм Евклида

Для быстрого вычисления наибольшего общего делителя двух чисел используют *алгоритм Евклида*. Он построен на следующем соотношении: НОД(a, b) = НОД$(a \bmod b, b)$.

Реализуйте рекурсивный алгоритм Евклида в виде функции `gcd(a, b)`.

Ввод	Вывод
12 16	4

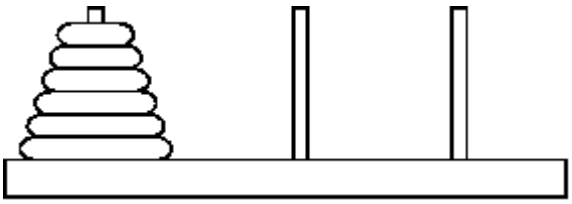
S: Ханойские башни

Головоломка “Ханойские башни” состоит из трех стержней, пронумерованных числами 1, 2, 3. На стержень 1 надета пирамидка из n дисков различного диаметра в порядке возрастания диаметра. Диски можно перекладывать с одного стержня на другой по одному, при этом диск нельзя класть на диск меньшего диаметра. Необходимо переложить всю пирамидку со стержня 1 на стержень 3 **за минимальное число перекладываний**.

Напишите программу, которая решает головоломку; для данного числа дисков n печатает последовательность перекладываний в формате `a b c`, где `a` — номер перекладываемого диска, `b` — номер стержня с которого снимается данный диск, `c` — номер стержня на который надевается данный диск.

Например, строка `1 2 3` означает перемещение диска номер 1 со стержня 2 на стержень 3. В одной строке печатается одна команда. Диски пронумерованы числами от 1 до n в порядке возрастания диаметров.

Программа должна вывести минимальный (по количеству произведенных операций) способ перекладывания пирамидки из данного числа дисков.



Указание: подумайте, как переложить пирамидку из одного диска? Из двух дисков? Из трех дисков? Из четырех дисков? Пусть мы научились перекладывать пирамидку из n дисков с произвольного стержня

на любой другой, как переложить пирамидку из $n + 1$ диска, если можно пользоваться решением для n дисков.

Напишите функцию `move (n, x, y)`, которая печатает последовательнось перекладываний дисков для перемещения пирамидки высоты `n` со стержня номер `x` на стержень номер `y`.

Ввод	Вывод
2	1 1 2 2 1 3 1 2 3

T: Ремонт в Ханое

Постановлением ЮНЕСКО оригинал Ханойской башни был подвергнут реставрации. В связи с этим во время пользования головоломкой нельзя было перекладывать кольца с первого стержня сразу на третий и наоборот.

Решите головоломку с учетом этих ограничений. Вам не нужно находить минимальное решение, но количество совершенных перемещений не должно быть больше 200000, при условии, что количество дисков не превосходит 10.

Тесты к этой задаче закрытые.

Ввод	Вывод
1	1 1 2 1 2 3
2	1 1 2 1 2 3 2 1 2 1 3 2 1 2 1 2 2 3 1 1 2 1 2 3

U: Циклические башни

На дорогах Ханоя было введено одностороннее круговое движение, поэтому теперь диск со стержня 1 можно перекладывать только на стержень 2, со стержня 2 на 3, а со стержня 3 на 1.

Решите головоломку с учетом этих ограничений. Вам не нужно находить минимальное решение, но количество совершенных перемещений не должно быть больше 200000, при условии, что количество дисков не превосходит 10.

Ввод	Вывод
1	1 1 2 1 2 3
2	1 1 2 1 2 3 2 1 2 1 3 1

	2 2 3 1 1 2 1 2 3

V: Несправедливые башни

В Ханое несправедливо запретили класть самый маленький диск (номер 1) на средний колышек (номер 2).

Решите головоломку с учетом этих ограничений. Вам не нужно находить минимальное решение, но количество совершенных перемещений не должно быть больше 200000, при условии, что количество дисков не превосходит 10.

Ввод	Вывод
2	1 1 3 2 1 2 1 3 1 2 2 3 1 1 3

W: Сортирующие башни

Первоначально все диски лежат на стержне номер 1. Переместите диски с нечетными номерами на стержень номер 2, а с четными номерами - на стержень номер 3.

Вам не нужно находить минимальное решение, но количество совершенных перемещений не должно быть больше 200000, при условии, что количество дисков не превосходит 10.

Ввод	Вывод
2	1 1 2 2 1 3
3	1 1 2 2 1 3 1 2 3 3 1 2 1 3 2

X: Обменные башни

Как и в предыдущих задачах, дано три стержня, на первом из которых надето *n* дисков различного размера. Необходимо их переместить на стержень 3 по следующим правилам:

Самый маленький диск (номер 1) можно в любой момент переложить на любой стержень. Перемещение диска номер 1 со стержня `a` на стержень `b` будем обозначать `1 a b`.

Можно поменять два диска, лежащих на вершине двух стержней, если размеры этих дисков отличаются на 1. Например, если на вершине стержня с номером `a` лежит диск размером 5, а на вершине стержня с номером `b` лежит диск размером 4, то эти диски можно поменять местами. Такой обмен двух дисков будем обозначать `0 a b` (указываются номера стержней, верхние диски которых обмениваются местами).

Для данного числа дисков n, не превосходящего 10, найдите решение головоломки. вам не нужно находить минимальное решение, но количество совершенных перемещений не должно быть больше 200000.

Ввод	Вывод
1	1 1 3
2	1 1 3 0 1 3 1 1 3

Y: Фишки

Дана полоска из клеток, пронумерованных от 1 до N слева направо. Разрешено снимать или ставить фишку на клетку с номером 1 или на клетку, следующую за самой левой из установленных фишек. Изначально полоска пуста. Нужно разместить фишки во всех клетках.

Программа получает на вход количество клеток в полоске N ($1 \le N \le 10$).

Программа должна вывести последовательность номеров клеток, с которыми совершается действие. Если фишка снимается, то номер клетки должен выводиться со знаком минус. Количество действий не должно превышать 10^4. Если существует несколько возможных решений задачи, то разрешается вывести любое.

Тесты к этой задаче закрытые.

Ввод	Вывод
3	1 2 -1 3 1

Z: Небоскреб

В небоскребе n этажей. Известно, что если уронить стеклянный шарик с этажа номер p, и шарик разобьется, то если уронить шарик с этажа номер $p + 1$, то он тоже разобьется. Также известно, что при броске с последнего этажа шарик всегда разбивается.

Вы хотите определить минимальный номер этажа, при падении с которого шарик разбивается. Для проведения экспериментов у вас есть два шарика. Вы можете разбить их все, но в результате вы должны абсолютно точно определить этот номер.

Определите, какого числа бросков достаточно, чтобы заведомо решить эту задачу.

Программа получает на вход количество этажей в небоскребе n и выводит наименьшее число бросков, при котором можно всегда решить задачу.

Тесты к этой задаче закрытые.

Ввод	Вывод
4	2
20	6

Комментарий к первому примеру. Нужно бросить шарик со 2-го этажа. Если он разобьется, то бросим второй шарик с 1-го этажа, а если не разобьется - то бросим шарик с 3-го этажа.

Подсказки.
1. Как следует действовать, если шарик был бы только один?
2. Пусть шариков два и мы бросили один шарик с этажа номер k. Как мы будем действовать в зависимости от того, разобьется ли шарик или нет?
3. Пусть `f(n)` - это минимальное число бросков, за которое можно определить искомый этаж, если бы в небоскребе было `n` этажей. Выразите `f(n)` через значения `f(a)` для меньших значений `a`.