Синтаксис Python. Функции, рекурсия.

Справочный материал можно найти в книжке (ссылка, страница 47.)
Задачи `A-H` требуется оформить следующим образом: написать функцию, название которой указано в условии, принимающую и возвращающую в точности описанные значения.
После описания функции и двух пустых строк написать строчку:
`exec(open('input.txt').read())`

Например, если задача состоит в написании функции, возвращающей сумму двух чисел, оформление решения будет выглядеть так:

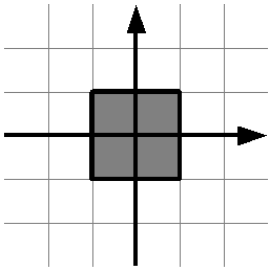
```
def sum_two_numbers(x, y):
    return x + y


exec(open('input.txt').read())
```

A. *Длина отрезка*
Даны четыре вещественных числа: x_1, y_1, x_2, y_2. Напишите функцию `Distance(x1, y1, x2, y2)`, принимающую на вход четыре вещественных числа и возвращающую расстояние между точкой (x_1, y_1) и (x_2, y_2).

Input	Output
`print(Distance(0.0, 0.0, 1.0, 1.0))`	`1.4142135623730951`

B. *Принадлежит ли точка квадрату - I*
Даны два вещественных числа x и y. Проверьте, принадлежит ли точка с координатами (x, y) заштрихованному квадрату (включая его границу). На рисунке сетка проведена с шагом 1.

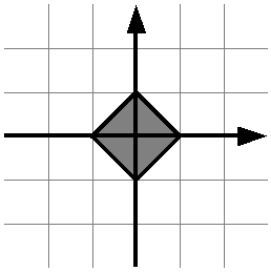


Решение должно содержать функцию `IsPointInSquare(x, y)`, принимающую на вход два вещественных числа и возвращающую логическое значение `True`, если точка принадлежит квадрату и логическое значение `False`, если не принадлежит.
Функция `IsPointInSquare` не должна содержать инструкцию `if`.

Input	Output
`print(IsPointInSquare(0.0, 0.0))`	`True`
`print(IsPointInSquare(3.0, -7.0))`	`False`

C. *Принадлежит ли точка квадрату - II*

Решите аналогичную задачу для такого квадрата:



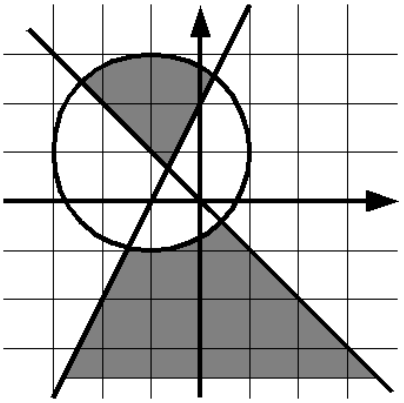
Решение должно содержать функцию `IsPointInSquare(x, y)`, принимающую на вход два вещественных числа и возвращающую логическое значение `True`, если точка принадлежит квадрату и логическое значение `False`, если не принадлежит.

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

Input	Output
`print(IsPointInSquare(0.0, 0.0))`	`True`
`print(IsPointInSquare(1.0, 1.0))`	`False`

D. *Принадлежит ли точка области*

Проверьте, принадлежит ли точка закрашенной серым цветом области (границы включаются). Центр окружности находится в точке $(-1, 1)$, её радиус равен 2. Прямые проходят через точки $(1, -1)$, $(-1, 1)$ и $(-1, 0)$, $(0, 2)$.



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

Решение должно соответствовать требованиям для решения задачи `B`.

Указание: уравнение окружности радиуса R с центром в точке (x_c, y_c):

$$(x - x_c)^2 + (y - y_c)^2 = R^2$$

Input	Output
`print(IsPointInArea(-4.0, -4.0))`	`False`
`print(IsPointInArea(-1.0, 2.0))`	`True`

E. *Минимальный делитель числа*

Дано натуральное число N $(1 < N < 10^9)$. Выведите его наименьший делитель, отличный от 1.

Решение оформите в виде функции `MinDivisor(x)`, принимающей на вход единственное натуральное число, бóльшее 1 и возвращающей его наименьший делитель, отличный от 1.

Алгоритм должен иметь сложность $O(\sqrt{N})$.

Указание: Если у числа N не нашлось делителя, не превосходящего $\sqrt{N}$, то число N — простое.

В решении нельзя использовать функцию `sqrt`.

Input	Output
`print(MinDivisor(4))`	2
`print(MinDivisor(9409))`	97
`print(MinDivisor(59))`	59

F. *Проверка числа на простоту*

Дано натуральное число N ($1 < N < 10^9$). Проверьте, является ли оно простым.

Решение оформите в виде функции `IsPrime(x)`, которая принимает на вход единственное натуральное число, бóльшее 1 и возвращает логическое значение `True` для простых чисел и логическое значение `False` для составных чисел.

Решение должно иметь сложность $O(\sqrt{N})$.

Input	Output
`print(IsPrime(4))`	`False`
`print(IsPrime(97))`	`True`

G. *Сумма делителей числа*

Для данного натурального числа N ($1 \leqslant N < 10^{10}$) требуется вычислить сумму его делителей, мéньших самого числа.

Решение оформите в виде функции `SumDivisors(x)`, принимающей на вход натуральное число и возвращающей сумму его делителей, отличных от него самого.

Решение должно иметь сложность $O(\sqrt{N})$.

Input	Output
`print(SumDivisors(12))`	`16`

H. *Дружественные числа*

Дружественные числа — это два различных натуральных числа, таких, что сумма всех делителей одного числа (меньших самого этого числа) равна другому числу, и наоборот (дружественными являются, например, 220 и 284).

Напишите функцию `IsFriend(x, y)`, которая проверяет пару чисел на «дружественность» и возвращает логическое значение `True`, если пара чисел дружественная и `False` в противном случае.

Функция `IsFriend` должна использовать функцию `SumDivisors` из предыдущей задачи.

Input	Output
`print(IsFriend(220, 284))`	`True`

I. *Дружественные числа в диапазоне*

Дружественные числа — это два натуральных числа, таких, что сумма всех делителей одного числа (меньших самого этого числа) равна другому числу, и наоборот (дружественными являются, например, 220 и 284).

Напишите программу, которая находит все пары не равных друг другу дружественных чисел в заданном диапазоне. Используйте функцию, которая вычисляет сумму делителей числа.

На вход программе подаётся две строки с натуральными числами a и b ($a < b$).

Программа должна вывести пары различных дружественных чисел, каждое из которых находится на отрезке $[a, b]$.

В каждой паре сначала выводится мéньшее число. Пары чисел должны выводиться в порядке возрастания первого числа из пары и разделяться запятой. Каждая пара заключена в скобки.

В случае, если таких пар в указанном диапазоне нет, вывести число 0.

Input	Output
1000 5000	(1184,1210) (2620,2924)

J. *Сумма цифр не меняется*

Напишите программу, которая находит все числа в диапазоне от a до b, сумма цифр которых не меняется при умножении на $2, 3, 4, 5, 6, 7, 8$ и 9 (например, число 9). Используйте функцию для вычисления суммы цифр числа.

На вход программе подаётся две строки с натуральными числами a и b ($a < b$).

Выведите числа, соответствующие условию задачи, в возрастающем порядке через пробел. Если на указанном отрезке таких чисел нет — ничего не выводите.

Input	Output
9 90	9 18 45 90

K. *Гиперпростые числа*

Простое число называется *гиперпростым*, если любое число, получающееся из него откидыванием нескольких последних цифр, тоже является простым. Например, число 733 — гиперпростое, так как и оно само, и числа 73 и 7 — простые. Напишите программу, которая определяет, верно ли, что переданное ей число N — гиперпростое.

Учтите, что число 1 не считается простым.

Input	Output
733	YES

L. *Гиперпростые числа в диапазоне*

Напишите программу, которая находит все гиперпростые числа в заданном диапазоне.

Если в указанном диапазоне гиперпростых чисел нет, вывести число 0.

Input	Output
30 100	31 37 53 59 71 73 79

■ Все задачи M–Y следует решить при помощи рекурсивных функций (разве что кроме задачи R).

Для изменения предельной глубины рекурсии используйте функцию `setrecursionlimit` из модуля `sys`. Глубина рекурсии — максимальное количество вызванных, но незаконченных (отложенных) функций в ходе выполнения программы. По умолчанию глубина рекурсии равна 1000. Пример использования:

```
from sys import setrecursionlimit
...описания функций
setrecursionlimit(200000)
...текст программы
```

M. *Сумма чисел*

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

Input	Output
1 2 3 0	6

N. *Разворот последовательности*

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

Решение этой задачи при помощи рекурсии позволяет обойтись без списков, строк и прочих структур данных для сохранения всех введённых чисел.

Input	Output
1 2 3 0	0 3 2 1

O° *Алгоритм Евклида*

Даны два неотрицательных целых числа A и B. Требуется найти их наибольший общий делитель при помощи уже известного вам алгоритма Евклида.

Решение будет принято, только если рекурсивная функция, вычисляющая НОД, будет иметь ОДИН условный оператор.

На вход программе подаётся два целых неотрицательных числа. Программа должна вывести одно число — ответ на вопрос задачи.

Input	Output
12 16	4

P. *Наименьшее общее кратное*

Наименьшее общее кратное (НОК) двух натуральных чисел —- это наименьшее число, которое делится нацело на оба исходных числа. Напишите программу, которая вычисляет НОК двух данных натуральных чисел.

Input	Output
12 16	48

Q° *Быстрое возведение в степень*

Дано вещественное число $a \neq 0$ и неотрицательное целое n ($n \leqslant 10^9$). Вычислите a^n.

Указание: воспользуйтесь тождествами $a^{2n} = (a^2)^n$ и $a^{2n+1} = a^{2n} \cdot a$

Input	Output
1.000000001 1000000000	2.7182820387553908

R* *Количество вызовов функции (числа Фибоначчи)*

Как известно, очередное число Фибоначчи равно сумме предыдущих двух. Первое и второе число Фибоначчи равны единице.

Программист Вася написал вычисление n-ого числа Фибоначчи с помощью рекурсивной функции, которая выглядит следующим образом:

```
def fibonacci(n):
    if n < 3:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

Сколько раз запустится эта функция прежде, чем будет получено значение?

На вход программе подаётся одно натуральное число. Программа должна вывести одно натуральное число — ответ на вопрос задачи. Гарантируется, что ответ не превосходит 10^{18}.

Input	Output
3	3
10	109
57	730870592323

S. *Сложение без сложения*

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

На вход программе подаются два целых неотрицательных числа a и b. Программа должна вывести их сумму $a + b$.

Input	Output
2 2	4

T. *Фишки*

Дана полоска из клеток, пронумерованных от 1 до N. На каждом ходе разрешено поставить фишку на клетку (если её там еще нет) или снять фишку с клетки (если она там есть). При этом можно выбрать не любую клетку, а только клетку под номером 1 или клетку с номером на 1 больше, чем у первой (слева) занятой клетки.

Изначально полоска пуста. Требуется занять все клетки.

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.

Input	Output
3	1 2 -1 3 1

U. *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 убывания диаметра (наверху самый маленький диск). Диски можно перекладывать с одного стержня на другой по одному, при этом диск нельзя класть на диск мéньшего 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.

Input	Output
2	1 1 2 2 1 3 1 2 3

V* *Ремонт в Ханое*

Решите задачу `U` со следующим ограничением: запрещено перекладывать диски со стержня 1 на стержень 3 и наоборот.

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

Input	Output
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

W* *Циклические башни*

Решите задачу `U` со следующим ограничением: диск со стержня 1 можно перекладывать только на стержень 2, со стержня 2 на 3, а со стержня 3 на 1.

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

Input	Output
2	1 1 2 1 2 3 2 1 2 1 3 1 2 2 3 1 1 2 1 2 3

X* *Несправедливые башни*

Решите задачу `U` со следующим ограничением: запрещено класть самый маленький **диск** (номер 1) на **средний** колышек (номер 2).

Input	Output
2	1 1 3 2 1 2 1 3 1 2 2 3 1 1 3

Y* *Сортирующие башни*

Решите задачу `U` в такой формулировке: первоначально все диски лежат на стержне номер 1. Переместите диски с нечётными номерами на стержень номер 2, а с чётными номерами — на стержень номер 3.

Input	Output
3	1 1 2 2 1 3 1 2 3 3 1 2 1 3 2

Z* *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 совершённых перемещений не должно быть больше 200000.

Input	Output
2	1 1 3 0 1 3 1 1 3

ZA* *Небоскрёб*

В небоскрёбе n этажей. Известно, что если уронить стеклянный шарик с этажа номер p, и шарик разобъётся, то если уронить шарик с этажа номер $p+1$, то он тоже разобъётся. 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 небоскрёбе n и выводит наименьшее число бросков, при котором можно всегда решить задачу.

Input	Output
4	2
20	6

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

Подсказки:

(a) Как следует действовать, если шарик был бы только один?

(b) Пусть шариков два и мы бросили один шарик с этажа номер k. Как мы будем действовать в зависимости от того, разобъется ли шарик или нет?

(c) Пусть $f(n)$ — это минимальное число бросков, за которое можно определить искомый этаж, если бы в небоскрёбе было n этажей. Выразите $f(n)$ через значения $f(a)$ для меньших значений a.