Мы уже пробовали запускать функцию из другой функции и все работало. Ничто не мешает запустить из функции саму себя - тогда просто создастся новый экземпляр функции, которые будет выполнять те же команды. Такой процесс называется **рекурсией**.

> *Представим себе, что у нас есть миллиард человек (это будущие экземпляры функции), сидящих в ряд, и у каждого из них есть листочек для записи (это его локальная память). Нам нужно произносить числа и написать инструкцию для людей так, чтобы они в итоге сказали все числа из последовательности в обратном порядке. Пусть каждый из них будет записывать на своем листочке только одно число. Тогда инструкция для человека будет выглядеть так:*
>
> *1) Запиши названное число*
> *2) Если число не последнее - потереби следующего за тобой человека, пришла его очередь работать*
> *3) Когда следующий за тобой человек сказал, что он закончил – назови записанное число*
> *4) Скажи тому, кто тебя теребил (предыдущий человек), что ты закончил*

Рассмотрим задачу. Пусть задается последовательность натуральных чисел, заканчивающаяся нулем. Необходимо развернуть ее с помощью рекурсии.

```
def rec():
    n = int(input())
    if n != 0:
        rec()
        print(n)
rec()
```

Эта функция осуществляет действие (вывод числа) на рекурсивном спуске, т.е. после рекурсивного вызова.

Рассмотрим задачу, где действия выполняются как на рекурсивном подъёме, так и на рекурсивном спуске. Пусть дана последовательность, которая оканчивается нулём. Необходимо вывести все чётные члены последовательности в прямом порядке, а затем все нечётные члены последовательности в обратном порядке. Её решение будет выглядеть так:

```
def rec():
    n = int(input())
    if n != 0:
        if n % 2 == 0:
            print(n)
        rec()
        if n % 2 != 0:
            print(n)
rec()
```

Каждый экземпляр функции считывает в свою локальную переменную n число, если оно чётное, то сразу выводит его и запускает следующий экземпляр. После того, как все последующие экземпляры функции окончили работу (и вывели последующие нечётные числа в обратном порядке), функция выводит число, если оно было нечетным.

Рассмотрим еще один пример: подсчитать факториал числа, не пользуясь циклами:

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
n = int(input())
print(factorial(n))
```

Тем, кто знаком с методом математической индукции, будет довольно просто осознать рекурсию. Как и в математической индукции, в рекурсии должна быть база (момент, когда функция не вызывает другую рекурсивную функцию) и переход (правило, по которому считается результат по известному результату для меньшего параметра). Наша функция подсчета факториала делает только свою работу, но пользуется результатами чужого труда. Например, если функция получила на вход параметр 4, то должна вернуть 4 умноженное на 3! (который будет посчитан другими функциями). В случае факториала аналогом "базы индукции" может выступать 0! - по определению он равен единице.

Эти примеры иллюстрируют общую схему написания рекурсивных функций: сначала проверяется условие, когда функция должна закончиться, а дальше делается все остальное. При этом параметр должен сходиться к значению базы. Обычно это означает, что при каждом следующем вызове рекурсии параметр должен уменьшаться.