**Рекурсия**

**def** short\_story():

**print**("У попа была собака, он ее любил.")

**print**("Она съела кусок мяса, он ее убил,")

**print**("В землю закопал и надпись написал:")

short\_story()

Функция может вызывать другую функцию. Но функция также может вызывать и саму себя! Вызов функции самой себя называется  **рекурсией.**

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.

**def** factorial(n):

**if** n == 0:

**return** 1

**else**:

**return** n \* factorial(n - 1)

**print**(factorial(5))

Рекурсивные функции являются мощным механизмом в программировании. К сожалению, они не всегда эффективны.

Для того, чтобы реализовать рекурсию нужно ответить на следующие вопросы:

 Какой случай (для какого набора параметров) будет крайним (простым) и что функция возвращает в этом случае?

 Как свести задачу для какого-то набора параметров (за исключением крайнего случая) к задаче, для другого набора параметров (как правило, с меньшими значениями)?

При этом программирование рекурсии выглядит так. Функция должна сначала проверить, не является ли переданный набор параметров простым (крайним) случаем. В этом случае функция должна вернуть значение (или выполнить действия), соответствующие простому случаю. Иначе функция должна вызвать себя рекурсивно для другого набора параметров, и на основе полученных значений вычислить значение, которое она должна вернуть.

**Проверка строки на палиндромность**

Строка является палиндромом, если она одинаково читается как справа налево, так и слева направо. Напишем функцию IsPalindrome, которая возвращает значение типа bool в зависимости от того, является ли строка палиндромом.

Крайнее значение — пустая строка или строка из одного символа всегда палиндром. Рекурсивный переход — строка является палиндромом, если у нее совпадают первый и последний символ, а также строка, полученная удалением первого и последнего символа является палиндромом.

**def** is\_palindrome(s):

**if** **len**(s) <= 1:

**return True**

**else:**

**return** s[0] == s[-1] **and** is\_palindrome(s[1:-1])

**Числа Фибоначчи**

Последовательность Фибоначчи задана так: F0=0, F(1)=1, при n>1 число Фибоначчи с номером  вычисляется как Fn = Fn+1 + Fn+2. Для рекурсивного вычисления чисел Фибоначчи достаточно аккуратно запрограммировать эти соотношения:

**def** Fib(n):

**if** n <= 1:

**return** n

**else**:

**return** Fib(n — 1) + Fib(n — 2)

**Быстрое возведение в степень**

Одним из полезных применений рекурсии является алгоритм быстрого возведения в степень. Если вычислять степень числа *an* при помощи простого цикла, то понадобится (*n-1)* умножение. Но можно использовать рекуррентные соотношения:

*an = an-1∙a*, при нечетном

*an = (an/2)2*, при четном .

Это позволяет записать алгоритм, который будет выполнять не более, чем 2\*log2*n* умножений:

**def** power(a, n):

**if** n == 0:

**return** 1

**elif** n % 2 == 1:

**return** power(a, n - 1) \* a

**else:**

**return** power(a, n // 2) \*\* 2

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 начале раздела.

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 получится бесконечная цепочка.

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