**Рекурсия**

Мы уже сталкивались с тем, что функция может вызывать другую функцию. Но функция также может вызывать и саму себя! В этом случае она будет называться рекурсивной.

Рассмотрим в качестве примера функцию, которая вычисляет факториал числа nn, то есть произведение чисел от 11 до nn. Факториал числа nn обозначается как n!n!, n!=1⋅2⋅…⋅nn!=1⋅2⋅…⋅n. Сначала напишем нерекурсивную реализацию этой функции:

**def** factorial(n):

 res = 1

 **for** i **in** range(2, n + 1):

 res \*= i

 **return** res

Эта функция будет работать верно для всех неотрицательных значений nn. При nn, равном 00 или 11, цикл не выполнится ни разу, и функция, как и положено, будет возвращать число 11.

Теперь напишем рекурсивную реализацию функции factorial(). Нам известно, что 0!=10!=1, 1!=11!=1. А как вычислить величину n!n! для большого nn? Если бы мы могли вычислить величину (n−1)!(n−1)!, то тогда легко вычислили бы n!n!, поскольку n!=n⋅(n−1)!n!=n⋅(n−1)!. Но как вычислить (n−1)!(n−1)!? Если бы мы вычислили (n−2)!(n−2)!, то смогли бы вычислить (n−1)!=(n−1)⋅(n−2)!(n−1)!=(n−1)⋅(n−2)!. А  как вычислить (n−2)!(n−2)!? Если бы... В конце концов, мы дойдём до величины 0!0!, которая равна 11. Таким образом, для вычисления факториала мы можем использовать значение факториала для меньшего числа, а если n=0n=0, то сразу выводить ответ, равный 11:

**def** factorial(n):

 **if** n == 0:

 **return** 1

 **else**:

 **return** factorial(n - 1) \* n

Давайте напишем рекурсивную функцию print\_n(), которая печатает все числа от 11 до nn для заданного числа nn. Такая функция не будет ничего возвращать, а результатом её действий будут напечатанные ею числа. Сначала приведём нерекурсивную реализацию с использованием цикла **for**:

**def** print\_n(n):

 **for** i **in** range(1, n + 1):

 **print**(i)

Теперь напишем рекурсивную реализацию. Для этого обратим внимание на то, что если n=1n=1, то нужно напечатать число 11. Если же n>1n>1, то нужно напечатать числа от 11 до n−1n−1 (а это умеет делать наша функция, если её вызвать с параметром n−1n−1), а затем напечатать число nn. Получается, что при рекурсивном способе написания мы можем обойтись без цикла:

**def** print\_n(n):

 **if** n == 1:

 **print**(1)

 **else**:

 print\_n(n - 1)

 **print**(n)

Давайте попробуем в этой функции переставить две последние строки местами:

**def** print\_n(n):

 **if** n == 1:

 **print**(1)

 **else**:

 **print**(n)

 print\_n(n - 1)

Что будет делать такая функция? Так как сначала мы будем печатать число nn, а потом вызывать функцию с параметром n−1n−1, то такая функция будет печатать числа от nn до 11, то есть печатать то же самое, но в обратном порядке. Обратим внимание на то, что момент остановки рекурсии можно сделать другим. Мы переставали делать рекурсивные вызовы при условии n=1n=1. Но можно делать остановку рекурсии при n=0n=0. Тогда при n=0n=0 нам не нужно вообще ничего печатать. Тогда получим вот такую реализацию нашей функции:

**def** print\_n(n):

 **if** n > 0:

 print\_n(n - 1)

 **print**(n)

Важно помнить, что при разработке рекурсивной функции необходимо прежде всего оформлять условие завершения рекурсии и думать, почему рекурсия когда-либо завершит работу. Если же у нас нет случая, при котором рекурсия прекращается, или этот случай не всегда достигается, то мы получим бесконечную рекурсию.

**Быстрое возведение в степень**

Рассмотрим задачу о возведении числа в целую неотрицательную степень. Напишем рекурсивную функцию, которая решает эту задачу:

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

 **if** n == 0:

 **return** 1

 **else**:

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

Такая функция будет работать очень долго, если степень nn является большим числом. Например, для n=100n=100 такая функция сделает 100100 вызовов самой себя. Чтобы вычислить степень числа быстрее, можно использовать следующее наблюдение. Для того чтобы вычислить a100a100, можно сначала вычислить a50a50, а затем возвести результат в квадрат. Аналогично, для вычисления a50a50 нам потребуется a25a25. Вычислить a25a25 таким же способом не получится, так как число 2525 — нечётное. Поэтому для вычисления a25a25 используем значение a24a24, как мы это делали в предыдущей реализации. В итоге получим вот такую рекурсивную функцию:

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

 **if** n == 0:

 **return** 1

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

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

 **else**:

 a2 = power(a, n // 2)

 **return** a2 \* a2

Можно заметить, что такая реализация будет работать значительно быстрее. Так, для n=100n=100 наша функция уже не 100100 раз, а только 99 раз вызовет саму себя: a50a50, a25a25, a24a24, a12a12, a6a6, a3a3, a2a2, a1a1, a0a0.

**Рекурсия. Подключение стандартных модулей**

В языке Python имеется ограничение на максимальную глубину рекурсии. Допустим, мы вызвали функцию для вычисления факториала числа nn. Эта функция вызовет себя с параметром n−1n−1, а внутри этого вызова будет вызвана функция с параметром n−2n−2, и так пока значение параметра не дойдёт до значения 00. Таким образом, возникнет цепочка рекурсивных вызовов, которая уходит на глубину nn. Если, например, n=1000n=1000, то мы получим ошибку исполнения, связанную с превышением максимально допустимой глубины рекурсии.

Внутри языка Python есть счётчик, который хранит глубину рекурсии. Если значение глубины рекурсии превышает максимально допустимое значение, то программа получает ошибку во время исполнения. Это сделано для того, чтобы избежать ошибок, связанных с условием завершения рекурсии, которые приводят к бесконечной рекурсии и переполнению памяти компьютера. Бывают ситуации, когда ограничение на максимальную допустимую глубину рекурсии требуется увеличить. Например, если мы хотим вычислить факториал числа больше 10001000 с использованием рекурсивной функции. Для этого есть специальная функция в языке Python, которая называется setrecursionlimit(). Для того чтобы её использовать, требуется подключить модуль sys, в котором она лежит. Чтобы увеличить максимально допустимую глубину рекурсии до миллиарда, надо написать:

**import** sys

sys.setrecursionlimit(10 \*\* 9)

В этом примере мы подключили модуль sys и после этого использовали функцию setrecursionlimit() из этого модуля, при этом мы должны указывать перед именем функции название модуля и разделять эти имена точкой: sys.setrecursionlimit().

Однако, даже после этого мы можем получить ошибку выполнения программы. Дело в том, что у операционной системы есть своё ограничение на размер стека, которое может зависеть от различных факторов, и мы его превысили. Но нескольких миллионов глубина рекурсии достигать может.

В языке Python имеется много полезных модулей, которые можно подключить и использовать функции, которые в них имеются. Например, можно подключить модуль random для работы со случайными числами:

**import** random

Можно подключить модуль math, в котором лежат различные математические функции:

**import** math

В модуле math есть такие функции, как sqrt() для вычисления квадратного корня, factorial() для вычисления факториала числа, тригонометрические функции, экспонента, логарифм и многие другие.

Для копирования вложенных списков или других сложных объектов может понадобиться модуль copy:

**import** copy

Если мы подключили модуль math и хотим использовать функцию квадратного корня из этого модуля, то мы должны не забыть через точку указать название модуля перед названием функции:

math.sqrt(2)

С одной стороны, это удобно. Потому что мы можем написать свою функцию sqrt(), и это не приведёт к ошибке, так как функции sqrt() и math.sqrt записываются различным образом.

С другой стороны, не всегда хочется писать название модуля каждый раз перед вызовом функции. Чтобы этого не делать, можно использовать другой способ импортировать функции из модуля:

**from** math **import** sqrt

sqrt(2)

Можно импортировать сразу несколько функций из модуля:

**from** math **import** sqrt, factorial

**print**(sqrt(2), factorial(5))

Также можно импортировать сразу все функции из модуля следующей строчкой:

**from** math **import** \*