Лабораторная работа 4.

Рекурсия.

Рекурсия – мощный инструмент программирования, по выразительным возможностям близкий к циклам. Рекурсия широко применяется при решении игровых и переборных задач.

Рекурсия (от латинского recursio – возвращение) –это такой способ организации вычислений, при котором процедура или функция в ходе выполнения обращается сама к себе. Другими словами, рекурсия является методом определения функций (процедур), при котором определяемая функция применена в теле своего же собственного определения.

Классический пример рекурсивного определения понятия факториала N! (N≥0):

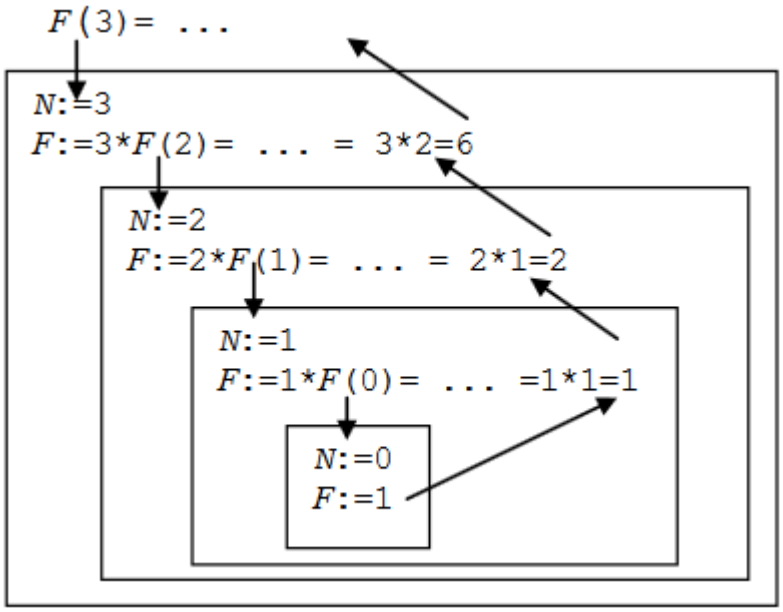
$$N! = \begin{cases} 1, & при\ N = 0 \\ N*(N-1)!, & при\ N > 0 \end{cases}$$

Здесь факториал N! выражается через факториал (N–1)! , т.е. через себя, поэтому данное определение N! является рекурсивным.

Пример реализации рекурсивной функции, решающей данную задачу:

```
long long f(int n)
{
    if (n==0) return 1;
    else return n*f(n-1);
}
```

Рассмотрим, как происходит выполнение рекурсивной функции на примере вычисления F(3)



Следует понимать, что при каждом новом вызове функции F заводится новая локальная переменная N, отличная от переменных N из других вызовов. Таким образом, одновременно существует несколько одноименных, но различных переменных. И если, например, переменной N из второго вызова присвоить какое-то значение, то это никак не скажется на значении переменной N из первого вызова.

Отметим, что максимальное количество активизированных и незавершённых вызовов одной и той же функции для фиксированного значения аргумента называется глубиной рекурсии. В нашем случае, при выполнении F(3) глубина рекурсии равна 4,что соответствует количеству вложенных прямоугольников на приведённом рисунке. Самый большой прямоугольник соответствует первому уровню рекурсии, а самый маленький – последнему, т.е. четвёртому уровню.

Уже на этом простом примере видна характерная особенность рекурсивной функции: при вычислении такой функции сначала мы постепенно упрощаем её аргумент, пока не дойдём до простейшего аргумента, при котором функция выдаёт явный ответ (без повторного обращения к самой себе), а затем мы начинаем двигаться в обратную сторону, вычисляя значения функции для всё более сложных аргументов.

В связи с этим описание рекурсивной функции должно состоять из двух частей. Первая часть –это не рекурсивные ветви, где рассматриваются простейшие случаи, для которых ответ дается явно. Отметим, что хотя бы одна не рекурсивная ветвь должна быть предусмотрена обязательно, иначе при вычислении функции мы никогда не выйдем из рекурсии (формально цепочка рекурсивных вызовов окажется бесконечно длинной). Вторая часть описания –это рекурсивные ветви, где рассматривается общий случай решения задачи. Общий случай сводится к более простым аналогичным случаям, для решения которых рекурсивно применяется та же самая функция, а затем из полученных ответов строится окончательный ответ.

Во всех представленных ниже задачах обязательно использовать рекурсивную функцию!

1. Дано натуральное число n. Выведите все числа от 1 до n.
2. Даны два целых числа A и B (каждое в отдельной строке). Выведите все числа от A до B включительно, в порядке возрастания/
3. Даны два целых числа A и B (каждое в отдельной строке). Выведите все числа от A до B включительно, в порядке убывания в противном случае.
4. В теории вычислимости важную роль играет функция Аккермана A(m,n), определенная следующим образом:

$$A(m,\ n) = \begin{cases} n+1, & m = 0; \\ A(m-1,\ 1), & m > 0,\ n = 0; \\ A(m-1,\ A(m,\ n-1)), & m > 0,\ n > 0. \end{cases}$$

Даны два целых неотрицательных числа m и n, каждое в отдельной строке. Выведите A(m,n).

5. **Точная степень двойки**

   Дано натуральное число N. Выведите слово YES, если число N является точной степенью двойки, или слово NO в противном случае. Операцией возведения в степень пользоваться нельзя!

6. **Сумма цифр числа**

   Дано натуральное число N. Вычислите сумму его цифр. При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется).

7. **Цифры числа справа налево**

   Дано натуральное число N. Выведите все его цифры по одной, в обратном порядке, разделяя их пробелами или новыми строками. При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется). Разрешена только рекурсия и целочисленная арифметика.

8. **Цифры числа слева направо**

   Дано натуральное число N. Выведите все его цифры по одной, в обычном порядке, разделяя их пробелами или новыми строками. При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется). Разрешена только рекурсия и целочисленная арифметика.

9. **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. Указание. Понятно, что задача сама по себе нерекурсивна, т.к. проверка числа n на простоту никак не сводится к проверке на простоту меньших чисел. Поэтому нужно сделать еще один параметр рекурсии: делитель числа, и именно по этому параметру и делать рекурсию.

10. **Разложение на множители**

    Дано натуральное число n>1. Выведите все простые множители этого числа в порядке неубывания с учетом кратности.

11. **Палиндром**

Дано слово, состоящее только из строчных латинских букв. Проверьте, является ли это слово палиндромом. Выведите YES или NO. При решении этой задачи нельзя пользоваться циклами, в решениях на питоне нельзя использовать срезы с шагом, отличным от 1.

12. **Разворот числа**

Дано число n, десятичная запись которого не содержит нулей. Получите число, записанное теми же цифрами, но в противоположном порядке. При решении этой задачи нельзя использовать циклы, строки, списки, массивы, разрешается только рекурсия и целочисленная арифметика.

Для 13 вариант необходимо выполнить задачу 1, для 14 варианту задачу 2 и так далее.