Рекурсия. Слово конечно страшное, но давайте попробуем с ним разобраться.
Давайте обратимся к Вики (в смысле Википедии, но если у Вас есть знакомая Вика, которая может Вам это объяснить, то я не возражаю :))
И так Реку́рсия — определение, описание, изображение какого-либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя. Термин «рекурсия» используется в различных специальных областях знаний — от лингвистики до логики, но наиболее широкое применение находит в математике и информатике.
С первого раза Вы точно ничего не поняли, поэтому прочитайте еще раз и не быстро, а потом еще раз, и еще раз (думаю 5 раз хватит).
Похоже на цикл правда? Ну так вот рекурсия это примерно и есть цикл. Читая определение рекурсии мы понимаем, что ничего не понимаем) и говорим себе - "Как дочитаю, прочту еще раз", то есть выполняя функцию, обращаемся в функции к себе же вызывая повтор. Давайте рассмотрим пример на известной Вам песне?

In [4]:

```
def ShortStory():
    print("У попа была собака, он ее любил.")
    print("Она съела кусок мяса, он ее убил,")
    print("В землю закопал и надпись написал:")
    ShortStory()
ShortStory()
# нет условий останова и нет шага рекурсии, результат ошибка
```

```
Она съела кусок мяса, он ее убил,
В землю закопал и надпись написал:
У попа была собака, он ее любил.
Она съела кусок мяса, он ее убил,
В землю закопал и надпись написал:
У попа была собака, он ее любил.
Она съела кусок мяса, он ее убил,
В землю закопал и надпись написал:
У попа была собака, он ее любил.
Она съела кусок мяса, он ее убил,
В землю закопал и надпись написал:
У попа была собака, он ее любил.
Она съела кусок мяса, он ее убил,
В землю закопал и надпись написал:
У попа была собака, он ее любил.
Она съела кусок мяса, он ее убил,
В землю закопал и надпись написал:
У попа была собака, он ее любил.
Она съела кусок мяса, он ее убил,
В землю закопал и надпись написал:
```

Как известно такую песню можно петь бесконечно. Примерно это и произойдет с Вашим кодом, пока это не надоест Python и он не выдаст ошибку). так же как если б вы читали про рекурсию не 5 раз, а бесконечность, Вам это надоест и Вы забросите чтение в любом месте, не факт, что дочитав определение до конца. В программе произойдет тоже самое, но скорее всего хуже.
Рассмотрим правильный подход к рекурсии.
Наиболее популярный пример правильной рекурсии, который Вы можете встретить в интернете это расчет математической функции факториала. (см. пример ниже)
Рекурсивные функции являются мощным механизмом в программировании. К сожалению, они не всегда эффективны. 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 выше. 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 получиться бесконечная цепочка.
Поэтому при разработке рекурсивной функции необходимо прежде всего оформлять условия завершения рекурсии и думать, почему рекурсия когда-либо завершит работу.

In [3]:

```
def factorial(n):
    if n <= 0: # условие останова
        return 1
    return n * factorial(n - 1) # шаг рекурсии
print ("Введите число для рассчета факториала")
n = input()
factorial(int(n))
```

```
Введите число для рассчета факториала
5
```

Out[3]:

```
120
```

Теперь, имея представление о том, как реализовать факториал давайте реализуем традиционную песенку в США и Канаде. Песенка часто поётся во время длительных поездок, поскольку у неё повторяющийся и легко запоминающийся мотив, а её пение может занять много времени. Особенно часто песню поют дети во время продолжительных групповых поездок на автобусе, таких как экскурсия школьного класса или выезд на природу бойскаутов "99 бутылок ". Помню эту песню первый раз услышал в одной из серий сериала "Альф". Найдите еще песню со счетом и реализуйте ее программно с использованием рекурсии. да, это пасхалка, как обычно проверяем вашу внимательность при прочтении.

Песню можно представить в следующем виде
<количество> бутылок пива на стене
<количество> бутылок пива!
Возьми одну, пусти по кругу
<количество минус 1> бутылок пива на стене!
останов рекурсии выполним при 0 бутылок на стене.
if n == 0:
        return 1
и не забудем уменьшать количество бутылок на стене (то есть шаг)
return song2 (n-1)

In [4]:

```
def song2(n):
    if n == 0: # условие останова
        return 1
    else:
        print (n, 'бутылок пива на стене')
        print (n, 'бутылок пива!')
        print ('Возьми одну, пусти по кругу')
        return  song2 (n-1) # шаг рекурсии
print ("Введите число для песни")
n = int(input())
song2(n)
```

```
Введите число для песни
5
5 бутылок пива на стене
5 бутылок пива!
Возьми одну, пусти по кругу
4 бутылок пива на стене
4 бутылок пива!
Возьми одну, пусти по кругу
3 бутылок пива на стене
3 бутылок пива!
Возьми одну, пусти по кругу
2 бутылок пива на стене
2 бутылок пива!
Возьми одну, пусти по кругу
1 бутылок пива на стене
1 бутылок пива!
Возьми одну, пусти по кругу
```

Out[4]:

```
1
```

Рассмотрим пример с нахождением суммы чисел от 1 до n
Если бы я хотел узнать сумму чисел от 1 до n, где n – натуральное число, я мог бы посчитать вручную 1 + 2 + 3 + 4 + ... + (несколько часов спустя) + n. А можно просто написать цикл for:

In [11]:

```
#решение через цикл
n = 4
for i in range (1, n):
    n += i
print(n)
```

```
10
```

In [12]:

```
#решение через рекурсию
def recursion(n):
    if n == 1: # условие останова
        return 1
    return n + recursion(n - 1) #шаг рекурсии
n=4
print(recursion(n))
```

```
10
```

У рекурсии есть несколько преимуществ в сравнении с первыми двумя методами. Рекурсия занимает меньше времени, чем выписывание 1 + 2 + 3 на сумму от 1 до 3. Для [рекурсии(4)] рекурсия может работать в обратную сторону:
Вызов функций: (4 -> 4 + 3 -> 4 + 3 + 2 -> 4 + 3 + 2 + 1 -> 10)
Принимая во внимание, что цикл [for] работает вперед: (1 -> 1 + 2 -> 1 + 2 + 3 -> 1 + 2 + 3 + 4 -> 10). Иногда рекурсивное решение проще, чем итеративное решение.

Итак можем сказать, что рекурсивная функция состоит из
1. Условие остановки или же Базовый случай
2. Условие продолжения или Шаг рекурсии – способ сведения задачи к более простым.

Вы также можете иметь «параллельные» рекурсивные вызовы функций. Например, рассмотрим последовательность Фибоначчи, которая определяется следующим образом:
Если число равно 0, то ответ равен 0.
Если число равно 1, то ответ равен 1.
В противном случае ответ представляет собой сумму двух предыдущих чисел Фибоначчи.

In [14]:

```
def fib(n):
    if n == 0 or n == 1: # базовый случай
        return n
    else:
        return fib(n - 2) + fib(n - 1) # шаг рекурсии
n=4
print(fib(n))
```

```
3
```

В данной практике решим пару, тройку простых задач на рекурсию, с целью понимания происходящего.
Задачи на рекурсию
1. Дано натуральное число n. Выведите все числа от 1 до n.
Ввод: 5
Вывод 1 2 3 4 5
2. 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!
Ввод: 8
Вывод: Yes
Ввод: 3
Вывод: No
3. Дано натуральное число N. Вычислите сумму его цифр. При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется).
Ввод: 234
Вывод: 9

4*. 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 и нельзя использовать срезы с шагом, отличным от 1
Пример: radar YES
Yes No
5. 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.
6*. 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.
7. Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом 0. Выведите все нечетные числа из этой последовательности, сохраняя их порядок.
В этой задаче нельзя использовать глобальные переменные и передавать какие-либо параметры в рекурсивную функцию. Функция получает данные, считывая их с клавиатуры. Функция не возвращает значение, а сразу же выводит результат на экран. Основная программа должна состоять только из вызова этой функции
8. Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом 0. Определите значение второго по величине элемента в этой последовательности, то есть элемента, который будет наибольшим, если из последовательности удалить наибольший элемент.
В этой задаче нельзя использовать глобальные переменные. Функция получает данные, считывая их с клавиатуры, а не получая их в виде параметра. В программе функция возвращает результат в виде кортежа из нескольких чисел и функция вообще не получает никаких параметров.
Гарантируется, что последовательность содержит хотя бы два числа (кроме нуля)
9. Дана последовательность натуральных чисел (одно число в строке), завершающаяся двумя числами 0 подряд. Определите, сколько раз в этой последовательности встречается число 1. Числа, идущие после двух нулей, необходимо игнорировать.
В этой задаче нельзя использовать глобальные переменные и параметры, передаваемые в функцию. Функция получает данные, считывая их с клавиатуры, а не получая их в виде параметров.
10. Даны числа a и b. Определите, сколько существует последовательностей из a нулей и b единиц, в которых никакие два нуля не стоят рядом
11. 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.
Фунция должна возвращать целое число, являющееся результатом работы программы, выводить число по одной цифре нельзя.
12* Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом 0. Выведите первое, третье, пятое и т.д. из введенных чисел. Завершающий ноль выводить не надо.
В этой задаче нельзя использовать глобальные переменные и передавать какие-либо параметры в рекурсивную функцию. Функция получает данные, считывая их с клавиатуры. Функция не возвращает значение, а сразу же выводит результат на экран. Основная программа должна состоять только из вызова этой функции. (два базовых случая)

In [ ]: