Лекция 12

Рекурсия в python

Что такое рекурсивные функции?

Рекурсия — это описание (определение) какого-либо объекта или процесса внутри этого самого объекта или процесса (структуры).

В Python механизм рекурсии очень распространен в функциях. Функциями называются участки кода, которые изолированы от остального кода программы. Их выполнение происходит только в том случае, если они вызываются. Все функции обладают одним общим свойством: они могут принимать параметры (ноль, один или несколько), а также возвращать значения (но это не всегда).

В языке Python, рекурсивная функция – **это функция, которая вызывает сама себя**, и при каждом очередном вызове использует данные, созданные во время предыдущего вызова.

Нужно сказать, что рекурсия подходит далеко не для каждой ситуации. Стоит учитывать такие факторы как:

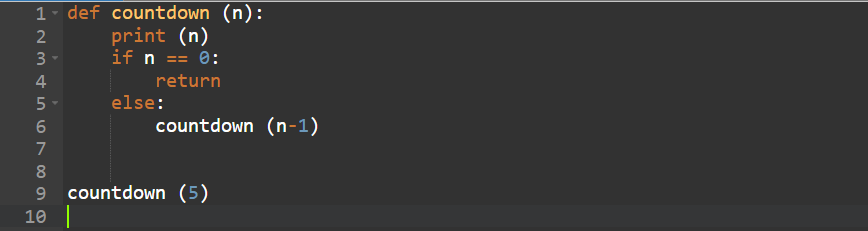
* Потребление памяти
* Время на выполнение
* Читаемость кода

Так же, стоит отметить что по умолчанию глубина рекурсии в языке Питон ограничена 1000 вызовов. Это ограничение можно поднять при помощи функции setrecursionlimit из модуля system.

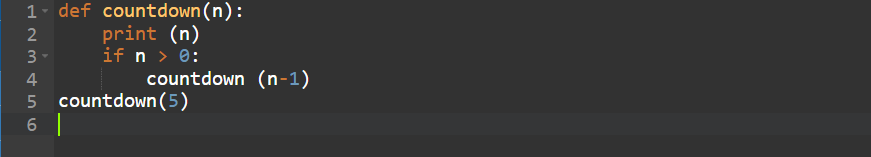
Примеры, представленные ниже, должны научить вас использовать рекурсию и помочь вам понять, когда вам следует выбрать её для решения ваших задач.

На примере countdown()

Первым примером является функция countdown(), которая принимает положительное число в качестве аргумента и выводит числа из указанного аргумента с уменьшением до нуля.



Версия countdown(), показанная выше, четко выделяет рекурсивный вызов, но есть более краткий способ его выразить.



Вывод идентичен в обоих случаях:

5

4

3

2

1

0

Стоит отметить countdown() не проверяет правильность своего аргумента. Если n является либо нецелым числом, либо отрицательным, вы получите исключение RecursionError, поскольку базовый регистр никогда не будет достигнут.

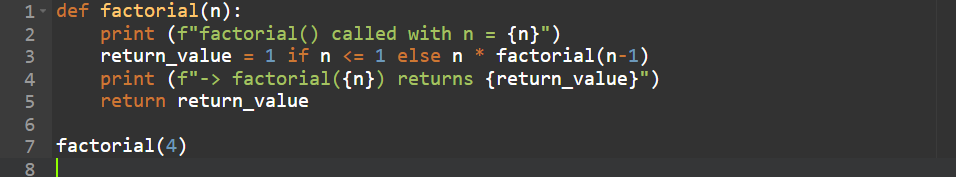
На примере факториала

Следующий пример связан с понятием факториала. Факториал целого положительного числа n, обозначается как n!, определяется следующим образом:

**n! = 1 \* 2 \* …\* n**

Другими словами, n! - это произведение всех целых чисел от 1 до n включительно.

Вот рекурсивная функция Python для вычисления факториала. Обратите внимание, насколько оно лаконично и насколько хорошо отражает приведенное выше определение (операторы print() дают более четкое представление о последовательности вызова и возврата).



Вывод:

factorial() called with n = 4

factorial() called with n = 3

factorial() called with n = 2

factorial() called with n = 1

-> factorial(1) returns 1

-> factorial(2) returns 2

-> factorial(3) returns 6

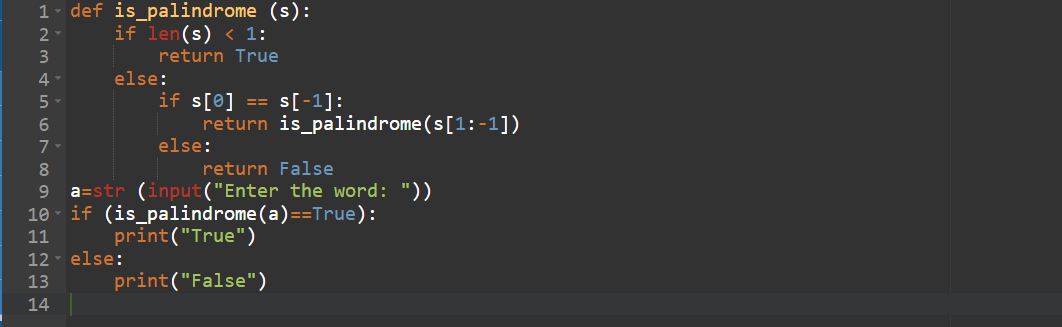
-> factorial(4) returns 24

24

Обратите внимание, как складываются все рекурсивные вызовы. Функция вызывается с n = 4, 3, 2 и 1 последовательно, прежде чем какой-либо из вызовов вернется. Наконец, когда n равно 1, проблема может быть решена без какой-либо дополнительной рекурсии. Затем каждый из сложенных рекурсивных вызовов разворачивается обратно, возвращая 1, 2, 6 и, наконец, 24 из самого внешнего вызова.

Выявить палиндром

В качестве последнего примера напишем программу, которая сможет определять, является ли введенная пользователем строка палиндромом.



В данном случае, требуются пояснения работы программы:

1. Пользователь вводит строку, которая записывается в переменную a.
2. Передаем переменную a в качестве аргумента в рекурсивную функцию is\_palindrome().
3. В качестве базового условия рекурсии принимаем значение длины строки меньше единицы. В этом случае выводим результат True.
4. В противном случае проверяем равенство первого и последнего элементов строки. Выбираем эти элементы при помощи индексов: s[0] == s[-1]. Если они равны, то опять вызываем нашу рекурсивную функцию, но передаем в нее в качестве аргумента строку без проверенных символов, используя для этого следующий срез строки: s[1:-1]. Если первый и последний элементы не равны, то условия основного свойства палиндрома нарушены, и мы выводим значение False.
5. Далее, в зависимости от результата работы функции, мы выводим ответ на вопрос нашей задачи. Если результат True, то строка является палиндромом, если False, то не является.

Вывод:

Enter the word:

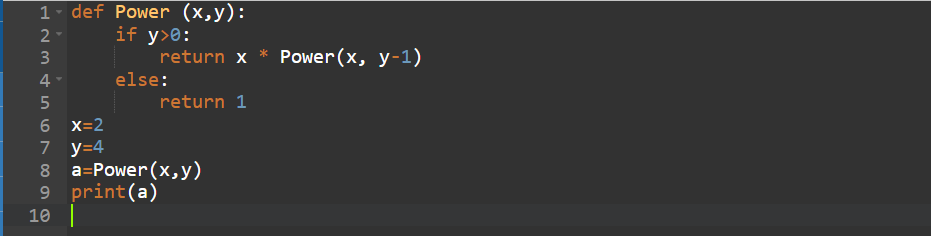
radar

True

Больше примеров

Так как рекурсивные функции в Python проще всего понять на примерах, рассмотрим ещё пару для закрепления.

Возведем число x в степень y с помощью функции Power(x, y). Рекурсивная функция Power(x, y) возвращает результат возведения действительного числа **x** в целую степень **y**. Предварительно предполагается, что значение y>0.



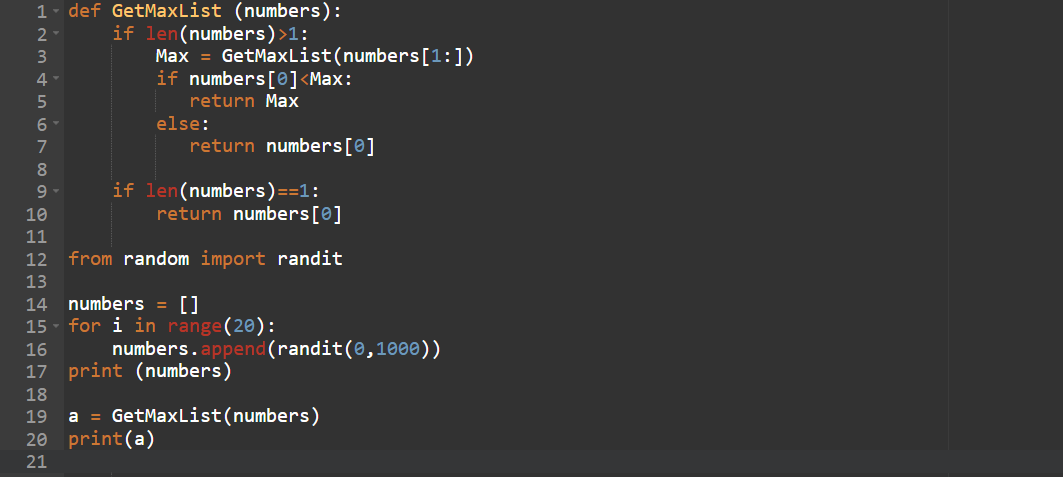
Вывод:

16

В качестве последнего примера используем функцию GetMaxList(), которая вычисляет максимальный элемент списка.

Вся работа функции разделена на 2 части:

1. выполнение действий, если в списке два и более чисел;
2. выполнение действий, если список заканчивается, то есть рассматривается последний элемент.



Вывод:

[591, 669, 985, 170, 256, 595, 563, 796, 891, 227, 889, 49, 404, 552, 272, 345, 641, 628, 520, 666]

985

Каждый рекурсивный вызов функции рассматривает список как две составляющие:

1. первый элемент списка (элемент с индексом 0);
2. следующие за первым элементы списка (элементы с индексами 1, 2, и т.д.).

Общий вывод

Благодаря поддержке механизма рекурсии, язык Python обеспечивает разработчика возможностью значительно повышать производительность приложений и экономии времени. Это делает Python универсальным и надежным средством для работы с производительными приложениями. Однако, 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. Поэтому стоит использовать рекурсивные функции осторожно.