Лекция 6.2. Понятие рекурсии и рекурсивных функции

Оглавление

[**Введение** 2](#_Toc180576516)

[**1.** **Рекурсия. Понятие и свойства** 2](#_Toc180576517)

[**2.** **Рекурсивные функции** 4](#_Toc180576518)

[Вычисление суммы натуральных чисел от 1 до n 6](#_Toc180576519)

[Проверка строки на палиндромность 6](#_Toc180576520)

[Суммирование списка 6](#_Toc180576521)

[Наибольшее значение в списке 6](#_Toc180576522)

[Числа Фибоначчи 7](#_Toc180576523)

[Быстрое возведение в степень 7](#_Toc180576524)

[Ханойские башни 8](#_Toc180576525)

[Ограничение на глубину рекурсии 9](#_Toc180576526)

[Детали работы рекурсивной функции 9](#_Toc180576527)

[Как работает рекурсия 10](#_Toc180576528)

[Рекурсивно или итеративно? 10](#_Toc180576529)

[**Заключение** 12](#_Toc180576530)

[**Список литературы** 12](#_Toc180576531)

**Алгоритмы поиска и сортировки**

**Цель:** формирование знаний в отношении природы и свойств рекурсии как явления в различных областях науки, в том числе в языке программирования Python, а также формирование знаний и умений в отношении программной реализации рекурсивных функций, области применения и способов использования данных функций при разработке программного кода на языке Python.

**План лекции:**Введение.1. Рекурсия. Понятия и свойства.  
2. Рекурсивные функции.  
Заключение.Список литературы.

# **Введение**

Рекурсия — это мощный и элегантный метод решения задач, при котором функция вызывает саму себя для разбиения проблемы на более мелкие и управляемые части. В языке Python рекурсия широко используется для обработки сложных структур данных, таких как деревья и графы, а также для решения задач, которые можно разделить на идентичные подзадачи, как, например, вычисление факториала или чисел Фибоначчи.

В этой статье мы рассмотрим, что такое рекурсия и как она реализуется в Python. Мы подробно разберем, как определить рекурсивную функцию, и обсудим важные аспекты, такие как условие завершения рекурсии, чтобы избежать бесконечных вызовов. Кроме того, мы рассмотрим примеры практического применения рекурсии и обсудим преимущества и недостатки использования рекурсивных подходов в программировании.

# **Рекурсия. Понятие и свойства**

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.

В математике: самый простой пример – ряд Фибоначчи. Первое и второе числа Фибоначчи равны 1. Для третьего и следующих элементов ряда Фибоначчи значение элемента равно сумме двух предыдущих элементов:

Ещё одним примером можно назвать прогрессии – арифметическую и геометрическую:

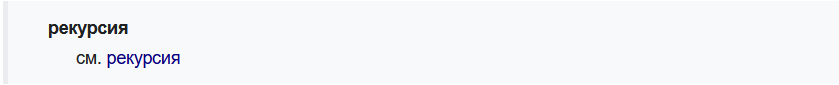
В физике: классическим примером бесконечной рекурсии являются два поставленные друг напротив друга зеркала: в них образуются два коридора из уменьшающихся отражений зеркал.



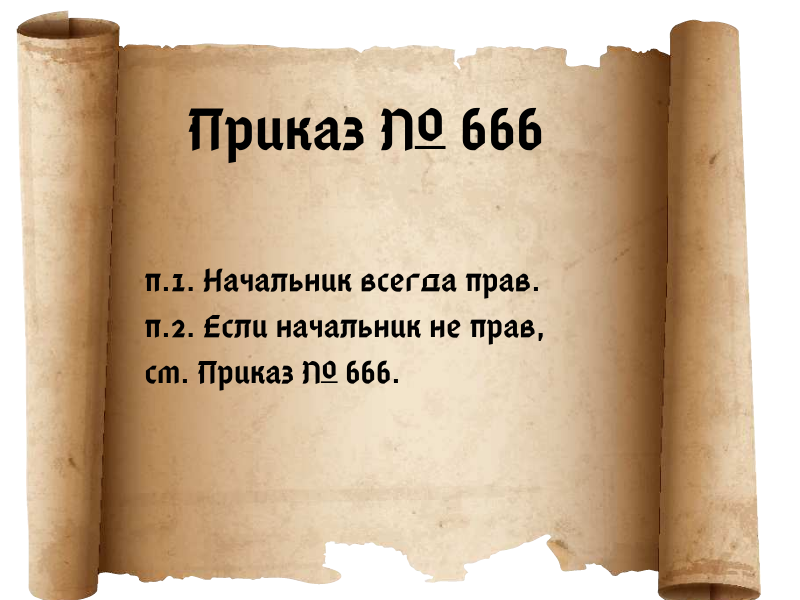
В лингвистике: рекурсией называют способность языка порождать вложенные предложения и конструкции. Базовое предложение «кошка съела мышь» может быть за счёт рекурсии расширено как «Ваня догадался, что кошка съела мышь», далее как «Катя знает, что Ваня догадался, что кошка съела мышь» и так далее. Рекурсия считается одной из лингвистических универсалий, то есть свойственна любому естественному языку.

В культуре: большая часть шуток о рекурсии касается бесконечной рекурсии, в которой нет условия выхода, например, известно высказывание: «чтобы понять рекурсию, нужно сначала понять рекурсию».

* Весьма популярна шутка о рекурсии, напоминающая словарную статью:



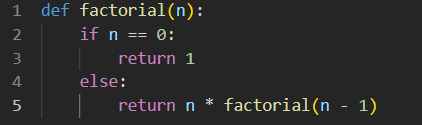
* Другая шутка о рекурсии — исправление запроса "рекурсия" на "рекурсия" в поисковой системе Google.
* Тема рекурсии присутствует во многих рассказах и очерках аргентинского писателя Хорхе Луиса Борхеса.
* Несколько рассказов Станислава Лема посвящены (возможным) казусам при бесконечной рекурсии:
  + рассказ из «Кибериады» о разумной машине, которая обладала достаточным умом и ленью, чтобы для решения поставленной задачи построить себе подобную и поручить решение ей (итогом стала бесконечная рекурсия, когда каждая новая машина строила себе подобную и передавала задание ей);
  + рассказ про Ийона Тихого «Путешествие четырнадцатое» из «Звёздных дневников Ийона Тихого», в котором герой последовательно переходит от статьи о сепульках к статье о сепуляции, оттуда — к статье о сепулькариях, в которой снова стоит отсылка к статье «сепульки».

В армии:

# **Рекурсивные функции**

Функции могут вызывать и другие функции, и даже вызывать сами себя! 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. Это можно сделать и в программе на Питоне:



Подобный прием (вызов функцией самой себя) называется рекурсией, а сама функция называется **рекурсивной.**

Как это работает? Допустим, мы вызвали функцию ***factorial(4).***

Будет вызвана функция, у которой значение параметра ***n= =4***. Она проверит условие ***n= =0***, поскольку условие ложно, то будет выполнена инструкция ***return n\*factorial(n-1)***. Но чтобы вычислить это значение, будет вызвана функция ***factorial(3)***, т. к. параметр ***n*** имеет значение, равное 4. Теперь в памяти будет находиться две функции ***factorial*** — одна со значением параметра ***n= =4***, а другая — ***n= =3***. При этом активна будет последняя функция.

Эта функция в свою очередь вызовет функцию ***factorial(2)***, та вызовет функцию ***factorial(1)***, затем ***factorial(0)***. В случае этой функции ничего более вызвано не будет, функция просто вернет значение **1**, и управление вернется в функцию ***factorial(1)***. Та умножит значение ***n= =1*** на значение **1**, которое вернула функция ***factorial(0)***, и вернет полученное произведение, равное **1**. Управление вернется в функцию ***factorial(2)***, которая умножит ***n= =2*** на значение **1**, которое вернула функция ***factorial(1)*** и вернет полученное произведение, равное **2**. Функция ***factorial(3)*** вернет **3\*2 = = 6**, а функция ***factorial(4)*** вернет **4\*6 = = 24**.

Таблица последовательности, в которой будут вызываться функции, приведена ниже. Значения функции возвращают в порядке, обратном порядке их вызова, то есть сначала заканчивает работу функция ***factorial(0)***, затем ***factorial(1)*** и т. д.

|  |  |
| --- | --- |
| С какими параметрами вызвана функция | Какое значение вернула |
| factorial(4) | 4 \* 6 == 24 |
| factorial(3) | 3 \* 2 == 6 |
| factorial(2) | 2 \* 1 == 2 |
| factorial(1) | 1 \* 1 == 1 |
| factorial(0) | 1 |

При отладке программы всю последовательность вложенных вызовов рекуррентных функций можно изучить в окне «Call Stack» («стек вызовов») в режиме отладки среды Wing IDE. При этом значения локальных переменных будут отображаться в окне «Stack data», и для каждой вызываемой функции значения локальных переменных будут своими.

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

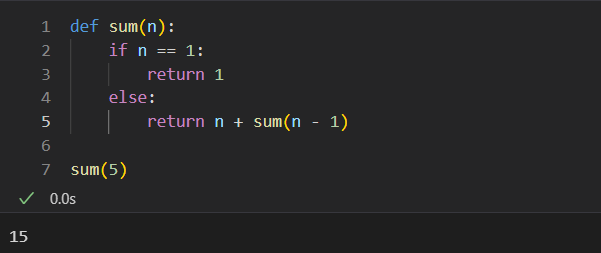
* Какой случай (для какого набора параметров) будет крайним (простым) и что функция возвращает в этом случае?
* Как свести задачу для какого-то набора параметров (за исключением крайнего случая) к задаче, для другого набора параметров (как правило, с меньшими значениями)?

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.

Рассмотрим несколько примеров (на самом деле эти примеры являются «учебными» и все перечисленные задачи гораздо лучше решать при помощи циклов, а не рекурсией).

## **Вычисление суммы натуральных чисел от 1 до n**

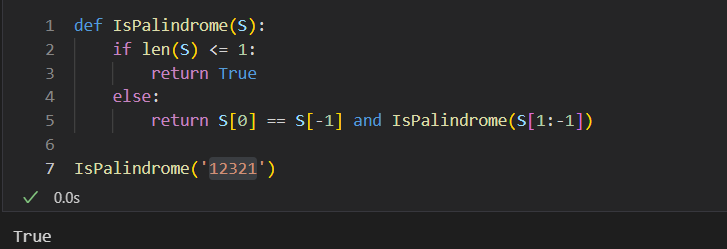
Если ***n*** = 1, то сумма равна 1. Иначе сумма чисел от 1 до ***n*** равна сумме чисел от 1 до ***n***-1, которую можно вычислить при помощи рекурсии плюс число ***n***.



## **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.



## **Суммирование списка**

Дан список чисел, необходимо просуммировать его.

Крайний случай — пустой список, сумма чисел в нем равна 0. Иначе нужно вычислить сумму чисел в срезе списка без одного элемента и добавить значение этого элемента.

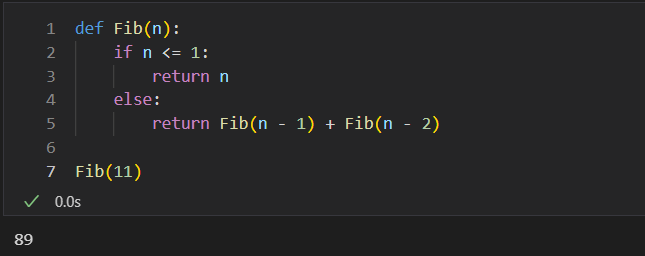
## **Наибольшее значение в списке**

Дан список чисел, необходимо найти наибольшее значение в нем.

Крайний случай — список из одного элемента, наибольшее значение в нем равно единственному элементу. Иначе нужно вычислить наибольшее значение в срезе списка без одного элемента и взять максимум из этого числа и значения этого элемента.

## **Числа Фибоначчи**

Последовательность Фибоначчи задана так: , при **n > 1** число Фибоначчи с номером **n** вычисляется как . Для рекурсивного вычисления чисел Фибоначчи достаточно аккуратно запрограммировать эти соотношения:

Рекурсивные функции являются мощным механизмом в программировании. К сожалению, они не всегда эффективны (об этом речь пойдет позже). 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 эпиграфе к этому разделу. Две наиболее распространенные причины для бесконечной рекурсии:

* Неправильное оформление выхода из рекурсии. Например, если мы в программе вычисления факториала забудем поставить проверку *if* *n==0*, то функция факториала вызовет функцию с аргументом ***-1***, и т.д.
* Рекурсивный вызов с неправильными параметрами. Например, если функция факториала с аргументом ***n*** будет вызывать функцию факториала с этим же аргументом, 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.

## **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*** умножений. Но можно использовать рекуррентные соотношения:

***an = an-1 \* a***, при нечетном ***n***

***an = (an/2)2 \* a***, при четном ***n***

Это позволяет записать алгоритм, который будет выполнять не более, чем ***2log2n*** умножений:

## **Ханойские башни**

Другой классической задачей, решаемой при помощи рекурсии, является задача о Ханойских башнях. Головоломка “Ханойские башни” состоит из трех стержней, пронумерованных числами 1, 2, 3. На стержень 1 надета пирамидка из n дисков различного диаметра в порядке возрастания диаметра. Диски можно перекладывать с одного стержня на другой строго по одному, при этом диск нельзя класть на диск меньшего диаметра. Необходимо переложить всю пирамидку со стержня 1 на стержень 3 за минимальное число перекладываний.

Необходимо написать программу, которая для данного числа дисков n печатает последовательность перекладываний, необходимую для решения головоломки. Решение данной задачи предлагается вам для самостоятельной работы



## **Ограничение на глубину рекурсии**

По умолчанию глубина рекурсии в языке Питон ограничена **1000** вызовов. Это ограничение можно поднять при помощи функции *setrecursionlimit* из модуля *system*. Например, чтобы увеличить возможную глубину рекурсии до 10000 нужно в начале программы выполнить две инструкции:

Следует учитывать, что глубину рекурсии нельзя увеличивать до очень большого значения, помимо ограничения, который устанавливается при помощи *setrecursionlimit* есть и ограничения операционной системы.

## **Детали работы рекурсивной функции**

Чтобы еще лучше понять, как это работает, разобьем на этапы процесс выполнения функции с параметром 3.

Для этого ниже представим каждый экземпляр с реальными числами. Это поможет «отследить», что происходит при вызове одной функции со значением 3 в качестве аргумента:

**# Первый вызов**

**factorial\_recursive(3):**

**if 3 == 1:**

**return 3**

**else:**

**return 3\*factorial\_recursive(3-1)**

**# Второй вызов**

**factorial\_recursive(2):**

**if 2 == 1:**

**return 2**

**else:**

**return 2\*factorial\_recursive(2-1)**

**# Третий вызов**

**factorial\_recursive(1):**

**if 1 == 1:**

**return 1**

**else:**

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

Рекурсивная функция не знает ответа для выражения 3\*factorial\_recursive(3–1), поэтому она добавляет в стек еще один вызов.

## **Как работает рекурсия**

**/\ factorial\_recursive(1) - последний вызов**

**|| factorial\_recursive(2) - второй вызов**

**|| factorial\_recursive(3) - первый вызов**

Выше показывается, как генерируется стек. Это происходит благодаря процессу LIFO (last in, first out, «последним пришел — первым ушел»). Как вы помните, первые вызовы функции не знают ответа, поэтому они добавляются в стек.

Но как только в стек добавляется вызов factorial\_recursive(1), для которого ответ имеется, стек начинает «разворачиваться» в обратном порядке, выполняя все вычисления с реальными значениями. В процессе каждый из слоев выпадает в процессе.

* **factorial\_recursive(1) завершается, отправляет 1 в**
* **factorial\_recursive(2) и выпадает из стека.**
* **factorial\_recursive(2) завершается, отправляет 2\*1 в**
* **factorial\_recursive(3) и выпадает из стека. Наконец, инструкция else здесь завершается, возвращается 3 \* 2 = 6, и из стека выпадает последний слой.**

Рекурсия в Python имеет ограничение в 3000 слоев.

## **Рекурсивно или итеративно?**

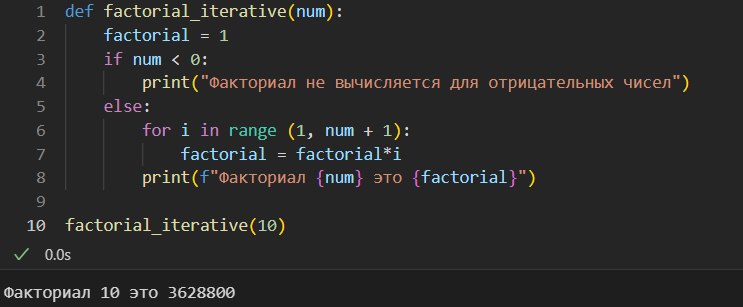
Каковы же преимущества рекурсивных функций? Можно ли с помощью итеративных получить тот же результат? Когда лучше использовать одни, а когда — другие?

Важно учитывать временную и пространственную сложности. Рекурсивные функции занимают больше места в памяти по сравнению с итеративными из-за постоянного добавления новых слоев в стек в памяти. Однако их производительность куда выше.

Рекурсия может быть медленной, если реализована неправильно

Тем не менее рекурсия может быть медленной, если ее неправильно реализовать. Из-за этого вычисления будут происходить чаще, чем требуется.

Написание итеративных функций зачастую требуется большего количества кода. Например, дальше пример функции для вычисления факториала, но с итеративным подходом. Выглядит не так изящно, не правда ли?



# **Заключение**

Рекурсия является важным инструментом в арсенале программиста, позволяя элегантно решать сложные задачи путем разбиения их на более простые подзадачи. В ходе этой статьи мы разобрали основные концепции рекурсии, изучили, как правильно определять и использовать рекурсивные функции в Python, а также рассмотрели примеры их применения.

Одним из ключевых аспектов рекурсии является наличие условия завершения, которое предотвращает бесконечные вызовы и обеспечивает корректное выполнение функции. Понимание этой концепции позволяет создавать более надежные и эффективные рекурсивные алгоритмы. Мы также обсудили преимущества рекурсии, такие как простота и читабельность кода для некоторых типов задач, и её потенциальные недостатки, такие как возможное превышение глубины стека вызовов.

Надеемся, что эта статья помогла вам лучше понять рекурсию и вдохновила на её использование в ваших собственных проектах. Рекурсивные функции могут значительно упростить решение определенных типов задач и расширить ваши возможности как разработчика, предоставляя мощный инструмент для работы с данными и алгоритмами.

# **Список литературы**

1. https://pythonru.com/osnovy/rekursiya-python

2. https://foxford.ru/wiki/informatika/rekursiya-v-python?utm\_referrer=https%3A%2F%2Fwww.google.com%2F