**56 Рекурсия. Базовые понятия и примеры**

**Рекурсией** называется способ определения множества объектов через это же множество на основе заданных базовых случаев. Возникает вопрос: «Как же множество объектов может определять само себя?».

Рассмотрим пример из математики. Числами Фибоначчи называется числовой ряд, в котором первые два числа равны единице, а все последующие являются суммой двух предыдущих.

Обратим внимание на то, что это определение состоит из двух частей. Вторая часть определения называется индуктивной, в ней число Фибоначчи определяется через два предыдущих числа, а первая часть определения указывает на те самые базовые случаи, то есть на то, что первые два числа Фибоначчи равны единице.

Использование рекурсии не ограничивается математикой и информатикой. Рассмотрим несколько примеров.

Так, англоязычная аббревиатура **GNU** – название бесплатной операционной системы с открытым исходным кодом – является рекурсивной. Она расшифровывается как «**GNU is not Unix**», что в переводе на русский язык означает «GNU не Unix».

Существует шутка, которая гласит о том, что для того, чтобы понять рекурсию, нужно сначала понять рекурсию. В сети Интернет популярны изобразительные примеры рекурсии.

**(Рис 56-1 - 56-9)** Также рекурсию содержит русская матрешка и даже герб Российской Федерации. В правой лапе двуглавый орел держит скипетр, увенчанный сверху уменьшенным изображением этого же герба, который, в свою очередь, также содержит скипетр и т. д.

Математические формы, известные как фракталы, принадлежат гению выдающегося ученого Бенуа Мандельброта. Большую часть жизни он провел в Соединенных Штатах, где преподавал математику в Йельском университете. В 1977 и 1982 годах Мандельброт опубликовал научные труды, посвященные изучению «фрактальной геометрии» или «геометрии природы», в которых разбивал на первый взгляд случайные математические формы на составные элементы, оказавшиеся при ближайшем рассмотрении повторяющимися, - что и доказывает наличие некого образца для копирования. Открытие Мандельброта возымело весомые позитивные последствия в развитии физики, астрономии и биологии. **(Рис. 56-10)**

Фрактал (от латинского «fractus» - разбитый, дробленый, сломанный) представляет собой сложную геометрическую фигуру, которая составлена из нескольких бесконечной последовательности частей, каждая из которых подобна всей фигуре целиком, и повторяется при уменьшении масштаба. **(Рис. 56-11, 56-12, 56-13)**

В природе фрактальными свойствами обладают многие объекты, например: кроны деревьев, цветная капуста, облака, кровеносная и альвеолярная системы человека и животных, кристаллы, снежинки, элементы которых выстраиваются в одну сложную структуру, побережья (фрактальная концепция позволила ученым измерить береговую линию Британских островов и другие, ранее неизмеримые, объекты) **(Рис. 56-14)**.

Фракталы широко применяются в компьютерной графике – при построении изображений деревьев, кустов, поверхности морей, горных ландшафтов, и других природных объектов. Благодаря фрактальной графике был изобретен эффективный способ реализации сложных неевклидовых объектов, чьи образы похожи на природные. **(Рис. 56-15 - 56-17)**

**Рекурсия в программировании**

|  |  |
| --- | --- |
| Как мы видели, функция может вызывать другую функцию. Но функция также может вызывать и саму себя!Напишем следующий модуль | **def short\_story(n):** **if n==0:** **print('...')** **return** **print("У попа была собака, он ее любил.")** **print("Она съела кусок мяса, он ее убил,")** **print("В землю закопал и надпись написал:")** **short\_story(n-1)****short\_story(5)** |

|  |  |
| --- | --- |
| Рассмотрим это на примере **функции вычисления факториала (Задача 3)**. 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: | **def factorial(n):** **if n==1:** **return 1** **else:** **return n \* factorial(n - 1)****print(factorial(5))****(Рис 56-18)** |

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

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. 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 == 1, то factorial(1) вызовет factorial(0), тот вызовет factorial(-1) и т. д.
2. Рекурсивный вызов с неправильными параметрами. Например, если функция factorial(n) будет вызывать factorial(n), то также получится бесконечная цепочка.

Поэтому при разработке рекурсивной функции необходимо прежде всего оформлять условия завершения рекурсии и думать, почему рекурсия когда-либо завершит работу.

**Задача 2 Числа Фибоначчи**

**Числа Фибоначчи** — элементы числовой [последовательности](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C)

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946,…

в которой каждое последующее число равно сумме двух предыдущих чисел. Название по имени средневекового математика Леонардо Пизанского (известного как [Фибоначчи](http://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%B1%D0%BE%D0%BD%D0%B0%D1%87%D1%87%D0%B8)).
Более формально, последовательность чисел Фибоначчи

 задается [линейным рекуррентным соотношением](http://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%B0%D1%8F_%D1%80%D0%B5%D0%BA%D1%83%D1%80%D1%80%D0%B5%D0%BD%D1%82%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C):[**](http://upload.wikimedia.org/math/9/4/3/9437eb760e953b06879cfb547f66b3f9.png)

Используя рекуррентное соотношение, можно построить рекурсивный алгоритм вычисления чисел Фибоначчи:

|  |  |
| --- | --- |
| **def fib(n):** **if n<3:** **return 1** **return fib(n-1) + fib(n-2)**Наибольший интерес в этом алгоритме представляет строчка 4:Интерес заключается в том, что дополнительный исполнитель действует по алгоритму — если его аргумент больше 1, он также вызывает очередного исполнителя для вычисления нужных ему значений. Получается серия вызовов, которые выстраиваются в *дерево рекурсивных вызовов*.  | http://1.bp.blogspot.com/-HqYpdwKjXXY/Ta2Dwxb3KxI/AAAAAAAAABc/JGmBainMIuI/s320/Untitled.jpg |

**Древовидно-рекурсивный процесс**

При вычислении значения *F*(5) будут вызваны процедуры вычисления *F*(4) и *F*(3). В свою очередь, для вычисления последних потребуется вычисление двух пар *F*(3), *F*(2) и *F*(2), *F*(1).
Можно заметить, что *F*(2) вычисляется три раза. Если рассмотреть вычисление *F*(*n*) при больших *n*, то повторных вычислений будет очень много. Это и есть основной недостаток рекурсии — повторные вычисления одних и тех же значений. Кроме того, с рекурсивными функциями связана одна серьезная ошибка: дерево рекурсивных вызовов может оказаться бесконечным и компьютер «зависнет». Важно, чтобы процесс сведения задачи к более простым когда-нибудь заканчивался. Для того, чтобы рекурсивный алгоритм заканчивал свою работу, необходимо, чтобы дерево рекурсивных вызовов при любых входных данных обрывалось и было конечным, т.е. должно существовать некое **терминальное условие**. В данном примере дерево рекурсивных вызовов обрывается на *F*1 и *F*2, для вычисления которых не используются рекурсивные вызовы.

В программировании рекурсия также применяется в виде рекурсивных алгоритмов. **Рекурсивными** называются вспомогательные алгоритмы, которые напрямую или через другие вспомогательные алгоритмы вызывают сами себя.

## Задача 6 Цифровой корень заданного натурального числа n

## Для вычисления цифрового корня числа само число заменяется суммой своих цифр до тех пор, пока числом не станет одна-единственная цифра.

Обозначим через **f** функцию цифрового корня из натурального числа, а через **g** – функцию суммы цифр числа.

В этой задаче мы можем определить, что **f (n) = n**, если число состоит из одной единственной цифры, то есть **n**< 10. В противном случае **f(n) = f (g (n))**.

Таким образом, нам остается решить задачу на вычисление суммы цифр натурального числа. Сумма цифр натурального числа равна самому числу, если оно состоит из одной цифры. То есть, если число меньше десяти, в противном случае сумма цифр числа равна сумме его правой цифры, выделенной как остаток от деления числа на 10, и суммы цифр оставшегося числа, полученного как число, деленное без остатка на 10.

( 125 8 976 4)

Как говорилось ранее, когда происходит вызов функции, компьютер сохраняет ее текущее состояние, то есть значение ее локальных параметров и адрес возврата. Эти данные сохраняются в особую область памяти – **стек**. Если забыть о программировании, то можно сказать, что стеком называется коробка с дном, но без крышки. Таким образом, любую вещь в эту коробку можно положить только поверх остальных, а извлечь из коробки можно только самую верхнюю вещь. При программировании стек работает аналогично. Он состоит из ячеек, а в процессоре есть специальный регистр, который называется указателем стека. В нем хранится адрес последней занятой ячейки стека. При первом вызове процедуры в стек записываются значения параметров основной программы, а также адрес возврата. При втором вызове функции происходит запись в стек значений локальных параметров при первом вызове, а также адрес возврата в него. При завершении очередного вызова функции из стека будут извлечены значения ее текущих параметров и адрес возврата.

Из этого следует, что при каждом вызове функции происходит расход не только оперативной, но и стековой памяти компьютера. Если в какой-то момент стековой памяти не хватит для исполнения программы, работа программы будет завершена **ошибкой переполнения программного стека**. Также при каждом вызове функции некоторое время занимает добавление данных в стек, поэтому часто **рекурсивные алгоритмы работают медленнее обычных итерационных**.

Вы наверняка обратили внимание на то, что обе предложенные задачи можно было решить без использования рекурсии, через циклы, но всегда ли так можно поступить? Всегда. Математиками доказано, что для решения любой задачи итерационный алгоритм можно заменить рекурсивным и, наоборот, рекурсивный алгоритм всегда можно заменить итерационным. Однако нет однозначного способа для того, чтобы сделать это для любой задачи. Также часто итерационный алгоритм получается куда сложнее рекурсивного. Однако, в силу причин, указанных ранее, **если задачу можно достаточно просто решить без использования рекурсии – лучше так и поступить**.

**Мы узнали:**

* **Рекурсией** называется способ определения множества объектов через это же множество на основе заданных базовых случаев.
* В программировании рекурсия применяется в виде **рекурсивных алгоритмов** – вспомогательных алгоритмов, которые напрямую или через другие вспомогательные алгоритмы вызывают сами себя.
* Рекурсивные алгоритмы обычно **короче и понятнее** итерационных, но при этом они работают медленнее и расходуют не только оперативную, но и стековую память. Поэтому, **если задачу можно достаточно просто решить, не используя рекурсию, – лучше так и поступить**.

**Задание 1**

Из каких частей состоит рекурсивное определение (выберите один или несколько вариантов из списка)?

|  |  |
| --- | --- |
|  |  Инверсная часть |
|  | Противоречащие случаи |
|  | Базовые случаи |
|  | Индуктивная часть |

 **Задание 2**

Написать функцию вычисления n – го элемента чисел Фибоначчи.

**Задание 3**

Написать функцию вычисления Факториала числа n.

**Задание 4**

|  |  |
| --- | --- |
| Чему будет равно а после исполнения инструкций? | **def func (num):****if num < 10:****return num****else:****return func (num % 10 + num % 100 //10)** **a = func (275)** |

**Задание 5**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Какие из приведенных строк кода можно описать на пропущенной позиции в рекурсивной функции вычисления **n**-ного числа Фибоначчи? Числами Фибоначчи называется числовой ряд, в котором первые два числа равны единице, а все последующие являются суммой двух предыдущих. Выберите один или несколько вариантов из списка | **def fibo (n):****if n <2:****return 1****else:****...** |

|  |  |
| --- | --- |
|  | return fibo (n - 2) + fibo (n - 1) |
|  | return fibo ((n - 1) + (n - 2)) |
|  | return fibo (n - 1) + fibo (n - 2) |
|  | fibo (n - 1) + fibo (n - 2) |

 |

**Задание 6**

Написать функцию вычисления Цифрового корня числа n.

**Задание 7**

Как называется область оперативной памяти, в которой хранятся значения параметров и адреса возврата вызываемых функций?

**Задание 8**

Что произойдет при попытке исполнения инструкций (выберите один вариант)?

**def rec (n):**

**if n > 0:**

**return rec (n % 10)**

**else:**

**return n**

 **a = rec (15)**

|  |  |
| --- | --- |
|  | Значение а станет равно 0 |
|  | Зацикливание |
|  | Переполнение программного стека |
|  | Значение а станет равно 5 |

**Задание 9**

Выберите **ложные** утверждения (один или несколько вариантов из списка).

|  |  |
| --- | --- |
|  | Рекурсивные алгоритмы обязательно должны содержать вызов самого себя или другого вспомогательного алгоритма. |
|  | Программный стек не ограничен |
|  | Рекурсивные алгоритмы обычно короче и понятнее итерационных. |
|  | Рекурсивные алгоритмы работают быстрее итерационных. |

**Задание 10 «Возведение в степень»**

Дано действительное положительное число a и целое неотрицательное число n. Вычислите a в степени n не используя циклы, возведение в степень через \*\* и функцию math.pow(), а используя рекуррентное соотношение a n=a⋅a n-1.

Решение оформите в виде функции power(a, n).

**Задание 11 «Разворот последовательности»**

Дана последовательность целых чисел, заканчивающаяся числом 0. Выведите эту последовательность в обратном порядке.

При решении этой задачи нельзя пользоваться массивами и прочими динамическими структурами данных. Рекурсия вам поможет.