Лабораторная работа №4. Рекурсивные функции. Графы.

Функции

Именные функции, инструкция def

Функция в python - объект, принимающий аргументы и возвращающий значение. Обычно функция определяется с помощью инструкции def.

Определим простейшую функцию:

def add(x, y):

return x + y

Инструкция return говорит, что нужно вернуть значение. В нашем случае функция возвращает сумму x и y.

Теперь мы ее можем вызвать:

>>> add(1, 10)

11

>>> add('abc', 'def')

'abcdef'

Функция может быть любой сложности и возвращать любые объекты (списки, кортежи, и даже функции!):

>>> def newfunc(n):

... def myfunc(x):

... return x + n

... return myfunc

...

>>> new = newfunc(100) # new - это функция

>>> new(200)

300

Функция может и не заканчиваться инструкцией return, при этом функция вернет значениеNone:

```
>>> def func():
... pass
...
>>> print(func())
None
```

Аргументы функции

Функция может принимать произвольное количество аргументов или не принимать их вовсе. Также распространены функции с произвольным числом аргументов, функции с позиционными и именованными аргументами, обязательными и необязательными.

```
>>> def func(a, b, c=2): # c - необязательный аргумент
... return a + b + c
...
>>> func(1, 2) # a = 1, b = 2, c = 2 (по умолчанию)
5
>>> func(1, 2, 3) # a = 1, b = 2, c = 3
6
>>> func(a=1, b=3) # a = 1, b = 3, c = 2
6
>>> func(a=3, c=6) # a = 3, c = 6, b не определен
Traceback (most recent call last):
File "", line 1, in
func(a=3, c=6)
TypeError: func() takes at least 2 arguments (2 given)
```

Функция также может принимать переменное количество позиционных аргументов, тогда перед именем ставится *:

```
>>> def func(*args):
... return args
```

...

>>> func(1, 2, 3, 'abc')

(1, 2, 3, 'abc')

>>> func()

()

>>> func(1)

(1,)

Как видно из примера, args - это кортежиз всех переданных аргументов функции, и с переменной можно работать также, как и с кортежем.

Функция может принимать и произвольное число именованных аргументов, тогда перед именем ставится **:

>>> def func(**kwargs):

... return kwargs

...

>>> func(a=1, b=2, c=3)

{'a': 1, 'c': 3, 'b': 2}

>>> func()

{}

>>> func(a='python')

{'a': 'python'}

В переменной kwargs у нас хранится словарь, с которым мы, опять-таки, можем делать все, что нам заблагорассудится.

Анонимные функции, инструкция lambda

Анонимные функции могут содержать лишь одно выражение, но и выполняются они быстрее. Анонимные функции создаются с помощью инструкции lambda. Кроме этого, их не обязательно присваивать переменной, как делали мы инструкцией def func():

>>> func = lambda x, y: x + y

>>> func(1, 2)

3

>>> func('a', 'b')

'ab'

>>> (lambda x, y: x + y)(1, 2)

3

>>> (lambda x, y: x + y)('a', 'b')

'ab'

lambda функции, в отличие от обычной, не требуется инструкция return, а в остальном, ведет себя точно так же:

>>> func = lambda *args: args

>>> func(1, 2, 3, 4)

(1, 2, 3, 4)

Понятие рекурсии, реализация в языке Python

В программировании рекурсия — вызов функции (процедуры) из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия), например, функция A вызывает функцию B, а функция B — функцию A. Количество вложенных вызовов функции или процедуры называется глубиной рекурсии.

Проще сказать нельзя. Про рекурсии есть известная поговорка:

Чтобы понять рекурсию, нужно сперва понять рекурсию

Итак, питон позволяет работать с рекурсиями легко и непринужденно. Самый первый пример рекурсии, с которой сталкиваются большинство программистов - это нахождение факториала. Код может быть таким:

```
def factorial(n):

    if n <= 1: return 1

    else: return n * factorial(n - 1)
```

Как видно, мы записали инструкцию if else слегка необычным для питона способом, но это позволяется в данном случаее, ввиду того, что читабельность здесь не ухудшается, но не следует злоупотреблять таким стилем. И вообще, PEP8всех рассудит. :)

Теперь проясним несколько важных особенностей, о которых всегда нужно помнить при работе с рекурсиями.

1. Существует ограничение на глубину рекурсии. По умолчанию оно равно 1000.
2. Для того, чтобы изменить это ограничение нужно вызвать функцию sys.setrecursionlimit(), а для просмотра текущего лимита sys.getrecursionlimit().
3. Не смотря на это - существует ограничение размером стека, который устанавливается операционной системой.
4. Рекурсия в Python не может использоваться в функциях-генераторах и сопрограммах. Однако, можно это поведение исправить, но лучше не стоит.
5. И последнее - применение декораторов к рекурсивным функциям может приводить к неожиданным результатам, поэтому будьте очень осторожны декорируя рекурсивные функции.

Также, всегда следует определять точки выхода из рекурсивных функций. Это как с циклами - бесконечный цикл может здорово «просадить» вашу операционную систему. И наконец, где лучше применять рекурсию, а где лучше воздержаться и обойтись, например циклами. Конечно, здесь многое зависит от задачи, но всегда следует помнить, что рекурсия в разы медленнее цикла. Так уж устроен питон, что вызов функции дорого вам обходится :) Вообще, в циклах не стоит вызывать функции, а уж рекурсивные функции и подавно.

Рекурсия хорошо подходит там, где производительность не слишком важна, а важнее читабельность и поддержка кода. К примеру, напишите две функции для обхода дерева каталогов, одну рекурсивную, другую с циклами.

Графы

Граф - это пара (V,E), где V - конечное непустое множество вершин, а E - множество неупорядоченных пар (u,v) вершин из V, называемых ребрами. Ребро s=(u,v) соединяет вершины u и v. Ребро s и вершина u (а также s и v) называются **инцидентными**, а вершины u и v - **смежными**. Степень вершины равна числу инцидентных ей ребер.

На рис. 11.1а приведен пример графа. В этом графе 7 вершин и 5 ребер.

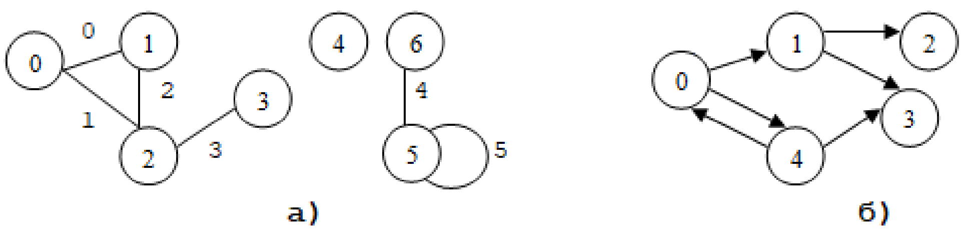

Рис. 11.1. Примеры графов: а) - неориентированный граф; б) - орграф

Ориентированный граф, или **орграф**, (V,E) отличается от обычного графа тем, что E - это множество упорядоченных пар (u,v) вершин, называемых дугами. Дуга (u,v) ведет от вершины ***u*** к вершине ***v***. Вершина ***u*** называется **предшественником *v***, а вершина ***v*** - **преемником *u***. Пример орграфа приведен на рис. 11.1б.

Графы представляются в программе чаще всего в виде **матрицы смежности** или **матрицы инцидентности**. На рис.11.2 приведен вид таких матриц для графа, изображенного на рис. 11.1а.

В матрице смежности 1 на пересечении i-й строки и j-го столбца означает, что вершины i и j смежны, а в матрице инцидентности - что вершина i и ребро j инцидентны. Если же ребро j является петлей вершины i, то элемент матрицы инцидентности g2[i][j] = 2.

g1 =

	0	1	2	3	4	5	6
0	0	1	1	0	0	0	0
1	1	0	1	0	0	0	0
2	1	1	0	1	0	0	0
3	0	0	1	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	1	1
6	0	0	0	0	0	1	0

а)

g2 =

	0	1	2	3	4	5
0	1	1	0	0	0	0
1	1	0	1	0	0	0
2	0	1	1	1	0	0
3	0	0	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	0	1	2
6	0	0	0	0	1	0

б)

Рис. 11.2. Примеры внутреннего представления графа:
а) – матрица смежности; **б)** - матрица инцидентности

Для орграфа элемент матрицы смежности g1[i][j] = 1, если есть дуга i –> j.

Для хранения графов удобно использовать списки (матрицы), но для хранения небольших графов можно задействовать словарь.

Внешнее представление графа может отличаться от внутреннего. Например, граф можно задать в виде количества вершин и последовательности ребер, где каждое ребро - пара смежных вершин:

```
7       количество вершин
0  1
0  2
1  2    ребра
2  3
6  5
5  5
```

(пример для графа, изображенного на рис. 11.1a)

Пример ввода такого графа с помощью словаря (+ вес ребра):

```
def read_graph():
    M = int(input()) # Количество рёбер
    G = {}
    for i in range(M):
        a, b, weight = input().split()
        a = int(a)  # Вершина А
        b = int(b)  # Вершина Б
        weight = int(weight)  # Вес ребра
        add_edge(G, a, b, weight)
        add_edge(G, b, a, weight)
    return G

def add_edge(G, a, b, weight):
    if a not in G:
        G[a] = {}
    G[a][b] = weight

G = read_graph()
print(G)
```

Пример результата работы программы:

```
6
0 1 1
0 2 1
1 2 1
2 3 1
6 5 1
5 5 1
{0: {1: 1, 2: 1}, 1: {0: 1, 2: 1}, 2: {0: 1, 1: 1, 3: 1}, 3:
{2: 1}, 6: {5: 1}, 5: {6: 1, 5: 1}}
>>>
```

Задание.

Задание 1. Задания по вариантам (варианты 1, 2, 3, 4).

1. Задан граф в виде количества вершин n<=10 и последовательности ребер (каждое ребро задается парой смежных вершин). Получить матрицу смежности.

а) Напечатать матрицу смежности. Проверить, есть ли в графе петли.

б) Напечатать матрицу смежности. Проверить, есть ли в графе вершины, не смежные с другими.

в) Напечатать для каждой вершины номера смежных вершин.

г) Проверить, есть ли в графе вершина, смежная со всеми другими вершинами.

д) Определить степень каждой вершины графа.

е) Напечатать номера вершин со степенью 1.

ж) Определить степень графа (максимальную степень его вершин).

2. Задан орграф в виде количества вершин n<=10 и последовательности дуг (дуга задается парой "предшественник преемник").

а) Напечатать номера вершин, имеющих более двух преемников.

б) Напечатать номера вершин, не имеющих предшественников.

в) Для каждой вершины напечатать номера всех предшественников.

г) Проверить, есть ли в графе вершины, имеющие только одного преемника.

3. Задан орграф в виде количества вершин n<=10 и матрицы смежности.

а) Напечатать номера вершин, имеющих и предшественников и преемников.

б) Напечатать список дуг орграфа в виде: v1 –> v2, где v1 – предшественник, v2 – преемник.

в) Напечатать номер вершины, имеющей наибольшее число преемников.

г) Определить число вершин, соединенных дугами в обоих направлениях.

4. Задан граф в виде количества вершин n<=7, количества ребер k<=28 и матрицы инцидентности.

а) Для каждой вершины напечатать список инцидентных ей ребер.

б) Определить степень каждой вершины графа.

в) Проверить, есть ли вершины со степенью 0.

г) Определить число вершин, инцидентных только одному ребру.

д) Определить наибольшее число смежных между собой ребер, инцидентных одной и той же вершине.

е) Проверить, есть ли в графе петли.

Задание 2. Выполните задание в соответствии с вашим вариантом.

1. Найти с заданной точностью корень функции $F(x)=x^2-2$ на отрезке (0,5) методом деления отрезка пополам.

2. Вычислить наибольший общий делитель двух целых чисел по алгоритму Евклида.

$$n,m>0,\ \mathrm{gcd}(m,n)=\begin{array}{ll} \mathrm{gcd}(n,m), & m<n; \\ n, & m=0; \\ \mathrm{gcd}(m-n,n), & m>n. \end{array}$$

3. Вычислить элементы последовательности:

$a(1)=1;$

$a(n)=n-a(a(n-1)),\ n>1;$

4. Вычислить значение полинома Лежандра порядка n в точке x:

$P_0(x)=1,$

$P_1(x)=x,$

$P_n(x)=(((2n-1)P_{n-1}(x)-(n-1)P_{n-2}(x))/n.$

5. Вычислить элементы последовательности:

$a(0)=1;$

$a(n)=a(n \text{ div } 2)+a(n \text{ div } 3),\ n>1;$