Занятие 7.

Рекурсия.

План занятия

- повторение функций
- рекурсивные функции
  - определение
  - необходимые атрибуты
  - задачи

Повторение изученного

Аргументы функции

In [4]:

```
def f(name, age, gender='male'):
    print(name, age)
```

In [5]:

```
f('Max', gender ='male', age=14)
```

```
Max 14
```

1-й аргумент позиционный, 2 и 3 - именованные

In [7]:

```
f('Max', age=14)
```

```
Max 14
```

$name$ аргумент позиционный, age именованный, $gender$ аргумент по умолчанию

In [6]:

```
f(gender='female', 'Anna', age=13)
```

```
  File "<ipython-input-6-4716d4427517>", line 1
    f(gender='female', 'Anna', age=13)
                       ^
SyntaxError: positional argument follows keyword argument
```

Сначала идут позиционные аргументы, затем именованные. Если пишешь именованный, то после него все аргументы именованные. Последовательность не важна.

In [9]:

```
f(gender='female', name='Anna', age=13)
```

```
Anna 13
```

In [12]:

```
f('Anna', 13, 'female')
```

```
Anna 13
```

Данные 2 команды эквивалентны

In [8]:

```
f(18, name='Ivan')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-4f03803dbef7> in <module>
----> 1 f(18, name='Ivan')

TypeError: f() got multiple values for argument 'name'
```

Несколько значений переменной $name$

In [13]:

```
f(18, 'Ivan')
```

```
18 Ivan
```

Ошибки синтаксиса нет, ошибка логическая.

Локальные и глобальные переменные

In [15]:

```
a = 10
def f(b):
    global a
    x = 15
    a += b
    print(a*x)
```

In [16]:

```
print(f(a))
```

```
300
None
```

Выводит None, т.к. функция f ничего не возвращает. Если изменить print на return, то будет выведено 300

Переменные x, b --- локальные, a --- глобальная

In [17]:

```
a = 10
def f(b):
    a = 7
    x = 15
    a += b
    print(a*x)
```

In [18]:

```
print(a)
```

```
10
```

Теперь переменная a также локальная

Если переменная внутри функции не объявляется и используется только для чтения (т.е. не изменяется), то она является **глобальной**. Если же она изменяется внутри функции, то она **локальная**.

In [21]:

```
def f(b):
    if b + a > 10:
        print('yep')

f(4)
```

```
yep
```

In [22]:

```
def f(b):
    a = b*2
    if b + a > 10:
        print('yep')

f(4)
```

```
yep
```

В первом случае, **a** глобальная, во втором - локальная

Рекурсия

Рекурсивная функция --- функция, которая

- вызывает саму себя (может быть косвенно)
- имет терминальное условие

In [1]:

```
def f(x):
    return f(x - 1)*2
```

In [2]:

```
def a(x):
    b(x)

def b(y):
    a(y)
```

В обоих примерах функции являются рекурсивными с точки зрения математики, но нерекурсивными с т.з. программирования, т.к. не имеют условия конца.

Факториал

In [1]:

```
def factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n-1)
```

In [4]:

```
print(factorial(1))
print(factorial(5))
print(factorial(10))
```

```
1
120
3628800
```

In [17]:

```
print(str(factorial(3000))[:20])
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
<ipython-input-17-f5df753832da> in <module>
----> 1 print(str(factorial(3000))[:20])
```

```
<ipython-input-3-444cef1f656d> in factorial(n)
      2     if n <= 1:
      3         return 1
----> 4     return n * factorial(n-1)

... last 1 frames repeated, from the frame below ...

<ipython-input-3-444cef1f656d> in factorial(n)
      2     if n <= 1:
      3         return 1
----> 4     return n * factorial(n-1)

RecursionError: maximum recursion depth exceeded in comparison
```

Превышение **максимальной глубины рекурсии**

In [3]:

```
import sys
print(f"Текущая максимальная глубина рекурсии: {sys.getrecursionlimit()}")
sys.setrecursionlimit(4000)
print(str(factorial(3500))[:20])
```

```
Текущая максимальная глубина рекурсии: 3000
23911281994776495250
```

In [ ]:

```
sys.setrecursionlimit(100000)
print(str(factorial(30000))[:20])
```

Stack over flow --- переполнение стека.

Следует различать превышение максимальной глубины рекурсии и переполнение стека.

In [6]:

```
def fact(n):
    product = 1
    for i in range(1, n+1):
        product *= i
    return product
```

In [8]:

```
print(str(fact(100000))[:20])
```

```
28242294079603478742
```

Вычисление чисел Фибоначчи

In [9]:

In [9]:

```
def fib(n):
    if n ==1 or n == 2:
        return 1
    return fib(n-1) + fib(n-2)
```

In [11]:

```
print(fib(5))
print(fib(10))
```

```
5
55
```

In [13]:

```
from time import time
begin = time()
print(fib(35))
print(f"time: {time() - begin}")
```

```
9227465
time: 2.2302708625793457
```

In [14]:

```
def fib_dynamic(n):
    a = 1
    b = 1
    if n <= 2:
        return 1
    for i in range(n - 2):
        a,b = a+b, a
    return a
```

In [15]:

```
from time import time
begin = time()
fib(35)
print(f"Recursion time: {time() - begin}")
begin = time()
fib_dynamic(35)
print(f"Dynamic time: {time() - begin}")
```

```
Recursion time: 2.3340442180633545
Dynamic time: 8.368492126464844e-05
```

Ханойские башни

Задача

Даны три стержня, на один из которых нанизаны n колец, причём кольца отличаются

размером и лежат меньшее на большем. Задача состоит в том, чтобы перенести пирамиду из n колец за наименьшее число ходов на другой стержень. За один раз разрешается переносить только одно кольцо, причём нельзя класть большее кольцо на меньшее.

Пусть **T(k)** --- количество шагов, необходимых, чтобы перетащить k дисков с одного стержня на другой. Тогда получим, что для перетаскивания n дисков нам нужно переместить сначала $n-1$ диск на промежуточный стержень (за $T(n-1)$ шаг), затем перетащить нижний диск на стержень 3, и за ним оставшиеся n-1.

$$T(n) = T(n-1) + 1 + T(n- 1) = 2T(n-1) + 1$$

Заметим, что $$T(1) = 1, T(2) = 3, T(3) = 7 \Rightarrow \boxed{T(n) = 2^n - 1}$$

In [16]:

```
def hanoi(n, l=1, r=3):
    t = 6 - l -r
    if n == 1:
        print(f'move from {l} to {r}')
        return
    hanoi(n-1, l, t)
    print(f'move from {l} to {r}')
    hanoi(n-1, t, r)
```

In [18]:

```
n = 3
hanoi(n)
```

```
move from 1 to 3
move from 1 to 2
move from 3 to 2
move from 1 to 3
move from 2 to 1
move from 2 to 3
move from 1 to 3
```

Проверить можно онлайн :)

http://hanoi.maramygin.ru/

Последовательности из 0 и 1

Вывести все последовательности, состоящие из 0 и 1 длины n.

In [20]:

```
def comb(n, s = ''):
```

```
    if n == 0:
        print(s)
        return
    comb(n-1, s + '0')
    comb(n-1, s + '1')
```

In [22]:

```
comb(3)
```

```
000
001
010
011
100
101
110
111
```

Эту же задачу можно решить с помощью встроенной функции **bin**

In [24]:

```
bin(13)[2:]
```

Out[24]:

```
'1101'
```

In [18]:

```
def f(n):
    for i in range(0, 2**n):
        print("0"*(n - len(bin(i)[2:])) + bin(i)[2:])
```

In [19]:

```
f(4)
```

```
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
```

Вывести все последовательности из {0,1,2,3,4}

In [25]:

```
def comb2(n, k, s = ''):
    if n == 0:
        print(s)
        return
    for i in range(k):
        char = str(i)
        comb2(n-1, k, s + char)
```

In [26]:

```
comb2(n=3, k=3)
```

```
000
001
002
010
011
012
020
021
022
100
101
102
110
111
112
120
121
122
200
201
202
210
211
212
220
221
222
```

Вывести все последовательности из 0 и 1 длины n, так что единиц ровно k

Простое решение

In [30]:

```
def comb_sum(n, k, s = ''):
```

```
    if n == 0:
        if s.count('1') == k:
            print(s)
        return
    comb_sum(n-1, k, s + '0')
    comb_sum(n-1, k, s + '1')
```

In [31]:

```
comb_sum(4,1)
```

```
0001
0010
0100
1000
```

Умное решение

In [32]:

```
def comb_sum(n, k, s = ''):
    if n == 0:
        print(s)

    if k > 0:
        comb_sum(n-1, k-1, s + '1')

    if n - k > 0:
        comb_sum(n-1, k, s + '0')
```

In [33]:

```
comb_sum(5,2)
```

```
11000
10100
10010
10001
01100
01010
01001
00110
00101
00011
```