Язык программирования Python

Процедуры и функции в языке Python

Вспомогательный алгоритм – это алгоритм решения какой-либо подзадачи, который может вызываться из основного алгоритма.

В программировании вспомогательные алгоритмы называют **подпрограммами**. В языке Python существуют два вида подпрограмм: процедуры и функции.

Процедура – это подпрограмма, которая выполняет некоторые действия после вызова её из основной программы или другой процедуры. Каждая процедура имеет уникальное ***имя,*** может иметь произвольное количество входных ***параметров***. При вызове процедуры указываются ***фактические значения параметров***.

Локальные переменные – это переменные, определённые в процедуре, они доступны только внутри процедуры.

Глобальные переменные – это переменные, определённые в основной программе. Они доступны внутри процедуры только для чтения, а для изменения требуется объявить их в процедуре после служебного слова **`global`**.

Описание процедуры

```
def <имя>(<параметры>):
    <операторы>
```

Процедура начинается служебным словом **def** (define – «определить»).
Формальные параметры процедуры перечисляются через запятую.
Операторы, входящие в тело процедуры, записываются с отступом.
Процедура должна быть определена до первого её вызова.

Вызов процедуры

Вызов процедуры осуществляется по её имени с указанием фактических параметров (аргументов).

```
<имя>(<аргументы>)
```

Примечание: Между **формальными** и **фактическими** параметрами должно быть соответствие по количеству, порядку следования и типу.

Работа процедуры

```
# пример процедуры
def summa(a, b):  # a, b - входные параметры
    global c      # глобальная переменная
    c = a+b       # сумма в глобальной переменной
```

```
# основная программа
summa(2, 3)    # вызов процедуры
print(c)           # напечатается число 5
. . .
```

При вызове процедуры её *формальные входные* параметры заменяются на *фактические*, значения *глобальных переменных* доступны в основной программе.

Примеры процедур

```
# процедура без параметра
def digit():
    print ("1111111")
# основная программа выводит
# три строки из семи единиц
digit()  # вызовы процедуры
digit()
digit()
```

```
1111111
1111111
1111111
```

Примеры процедур

```
# процедура с одним параметром
def digit(n):      # n - формальный параметр
    print ("1" * n)  # строка из n единиц
# основная программа выводит нужное количество единиц
digit(7)   # вызовы процедуры с фактическим параметром
digit(8)
digit(9)
```

```
1111111
11111111
111111111
```

Примеры процедур

```
# процедура с двумя параметрами
def digit(d, n):  # d, n - локальные переменные
    print (d * n) # строка из n цифр d
# основная программа
# выводит нужное количество заданных цифр
digit("1", 7) # вызовы процедуры с факт. параметрами
digit("2", 8)
digit("3", 9)
```

```
1111111
22222222
333333333
```

Функции

Функция – это вспомогательный алгоритм, который всегда возвращает в основной алгоритм значение-результат.

Описание функции

```
def <имя>(<параметры>):
    <операторы>
    return <результат>
```

После оператора **return** («вернуть») записывается результат, который возвращает функция. В функции может быть несколько операторов **return**, после выполнения любого из них работа функции заканчивается.

Вызов функции

Функции можно вызывать везде, где можно использовать выражение соответствующего типа (в операторах присваивания или вывода).

```
<имя>(<аргументы>)
```

Работа функции

```
# пример функции
def summa(a, b):  # a, b - параметры функции
    c = a+b       # вычисление функции
    return c      # возвращаемый результат

# основная программа
s = summa(2, 3)       # вызов функции
print(s)              # напечатается число 5
. . .
```

При вызове функции её *формальные* аргументы заменяются на *фактические*, по окончании выполнения значение функции передаётся в основную программу в место вызова.

Пример функции

```
# функция большее из двух
def max2(a, b):        # a, b - формальные параметры
    if a > b : m = a
    else:      m = b
    return m           # возвращаемый функцией результат
# основная программа
print(5, 2)
print("max =", max2(5, 2))  # вызов функции с аргументами
print(-2, 2)
print("max =", max2(-2, 2)) # вызов функции с аргументами
```

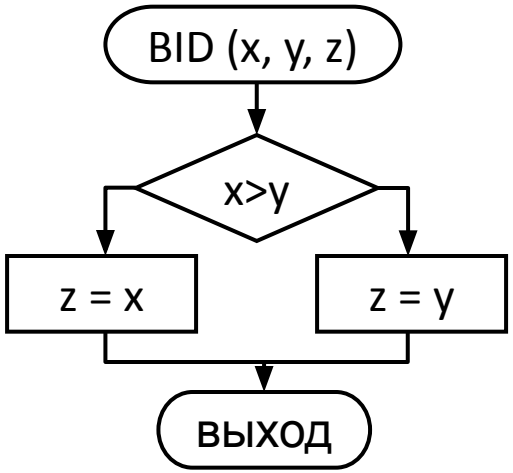
```
5 2
max = 5
-2 2
max = 2
```

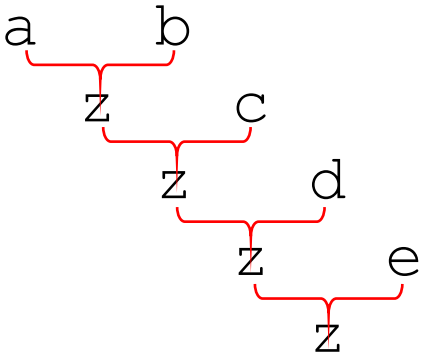
Примечание: В языке Python есть встроенная функция `max()`, вычисляющая максимальное значение из нескольких аргументов.

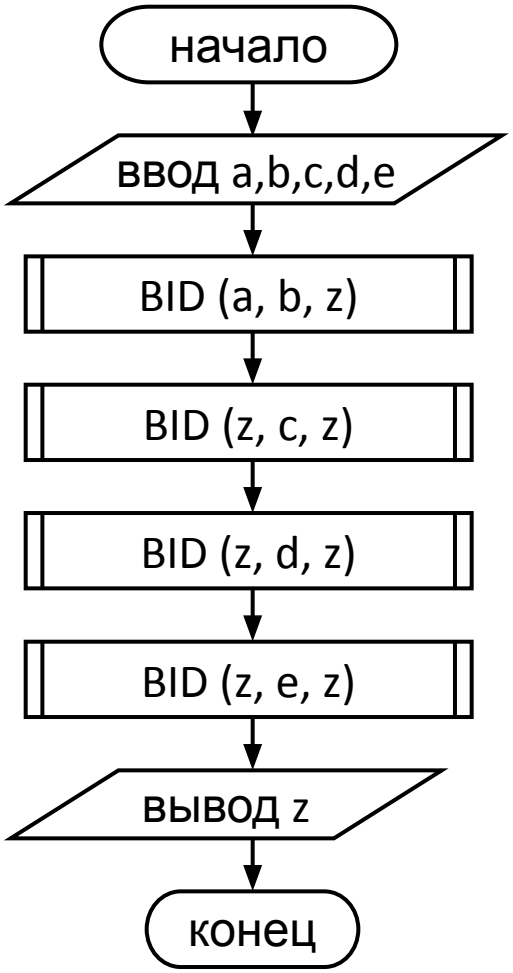
Задача 1а

Найти большее из пяти заданных чисел, используя вспомогательный алгоритм нахождения большего из двух чисел.

Блок-схема алгоритма решения задачи







Задача 1а

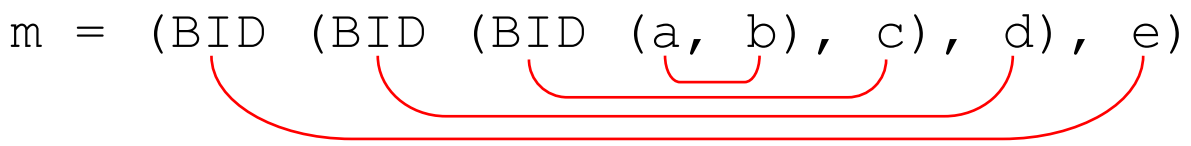
Найти большее из пяти заданных чисел, используя вспомогательный алгоритм нахождения большего из двух чисел.

```
# Большее из пяти чисел
def bid(x, y):   # Процедура большее из двух
    global z         # глобальная переменная
    if x>y: z=x
    else:   z=y
# Основная программа
print("Введите 5 чисел через пробел")
a,b,c,d,e = input().split()
a,b,c,d,e = int(a), int(b), int(c), int(d), int(e)
bid(a, b)
bid(z, c)
bid(z, d)
bid(z, e)
print("Максимальное число: ", z)
```

```
Введите 5 чисел через пробел
2 3 5 4 1
Максимальное число:  5
```

Задача 1б

Найти большее из пяти заданных чисел, используя вспомогательный алгоритм нахождения большего из двух чисел.




```
# Большее из пяти чисел
def bid(x, y):       # Функция большее из двух
    if x > y : z = x
    else:      z = y
    return z             # возвращаемый функцией результат
# Основная программа
print("Введите 5 чисел через пробел")
a,b,c,d,e = input().split()
a,b,c,d,e = int(a), int(b), int(c), int(d), int(e)
# вызовы функции в выражении
m = bid( bid (bid (bid (a, b), c), d), e)
print("Максимальное число: ", m)
```

```
Введите 5 чисел через пробел
2 5 1 3 4
Максимальное число:  5
```

Задача 2

Найти наибольший общий делитель чисел 16, 32, 40, 64, 80 и 128, используя в качестве процедуры алгоритм Евклида.

```
# Процедура НОД
def nod (a, b):       # a, b - формальные параметры
    global x          # объявление глобальной переменной
    while a != b:       # пока числа не равны
        if a > b:       # большее заменяем разностью чисел
            a = a-b
        else:
            b = b-a
    x = a           # результат процедуры НОД
# Основная программа
m = [16, 32, 40, 64, 80, 128] # массив исходных чисел
print (m)           # вывод чисел на экран
x = m[0]            # первое число для процедуры
for i in range (1, 6): # перебор чисел в массиве
    y = m[i]          # второе число для процедуры
    nod (x, y)        # вызов процедуры для этих чисел
print ("НОД = ", x)     # вывод результата
```

```
[16, 32, 40, 64, 80, 128]
НОД =  8
```

Задача 3

Вывести на экран запись целого числа (0 … 255) в 8-битном двоичном коде.

```
# Процедура перевод в двоичную систему
def print_bin (n):                  # n - целое число 0...255
    k = 128                         # делитель для старшей цифры
    while k > 0:                    # пока делитель больше 0
        d = n // k                  # очередная цифра
        print(d , end="")           # вывод очередной цифры
        n = n % k                   # заменяем число на остаток
        k = k // 2                  # делитель для следующей цифры
# Основная программа
x = int(input("Введите число 0...255: "))
print_bin (x)                       # вызов процедуры
```

```
Введите число 0...255: 85
01010101
```

Задача 4

Вывести на экран 10 первых членов последовательности Фибоначчи.
f(1) = 1, f(2) = 1, f(3) = f(1) + f(2), … , f(i) = f(i-2) + f(i-1)

```
# Функция вычисления n-го члена последовательности Фибоначчи
def f (n):
    if n==1 or n==2:          # первые два числа равны 1
        rez = 1
    else:                     # число равно сумме двух предыдущих
        rez = f(n-1) + f(n-2)  # рекурсивный вызов функции
    return rez
# Основная программа
for i in range(1, 11):     # для чисел от 1 до 10
    print (f(i), end=" ") # вывод очередного значения функции
```

```
1 1 2 3 5 8 13 21 34 55
```

Примечание: Функция, в которой происходит вычисление очередного значения функции через вычисление её предшествующих значений, называется **рекурсивной**.

Задача 5

Подсчитать количество слов в тексте, используя вспомогательный алгоритм нахождения количества пробелов в строке.

```
# Функция количество пробелов
def count_space(x):     # x - строка
    k = 0               # нач. знач. счетчика
    for c in x:         # перебор символов
        if c == " ":    # если пробел
            k = k+1     # увеличиваем счетчик
    return k            # результат функции
# Основная программа
a = input("Введите текст: ")
print("Количество слов в тексте", count_space(a)+1)
```

```
Введите текст: Мама мыла раму
Количество слов в тексте 3
```

Задача 6

Удалить все пробелы в тексте, используя вспомогательный алгоритм удаления символов в строке.

```
# Функция удаление символа
def del_char(x, y):     # x - строка, y - символ
    z = ""              # нач. знач. результата
    for c in x:         # перебор символов
        if c != y:      # если не заданный символ
            z = z+c     # присоединяем к результату
    return z            # результат функции
# Основная программа
a = input("Введите текст: ")
b = del_char(a, " ")
print("Изменённый текст:", b)
```

```
Введите текст: Мама мыла раму
Изменённый текст: Мамамылараму
```

Задача 7

Составить программу для вычисления числа сочетаний из *n* по *k*.

В комбинаторике набор ***k*** элементов, выбранных из данного множества, содержащего ***n*** различных элементов, называется сочетанием из ***n*** по ***k***. Значение этой величины вычисляется по формуле:

$$C_n^k = \frac{n!}{k!(n-k)!}$$

```
# Функция факториал
def fact(x):
    p = 1                           # нач. знач. произведения
    for i in range(1, x+1):         # для i от 1 до x
        p = p*i                     # добавить к произведению
    return p                        # результат функции
# Основная программа
n = int(input("Введите n (<13): "))
k = int(input("Введите k (<13): "))
C = fact(n) // fact(k) // fact(n-k)
print("Число сочентаний из", n, "по", k, "равно", C)
```

```
Введите n (<13): 7
Введите k (<13): 5
Число сочетаний из 7 по 5 равно 21
```