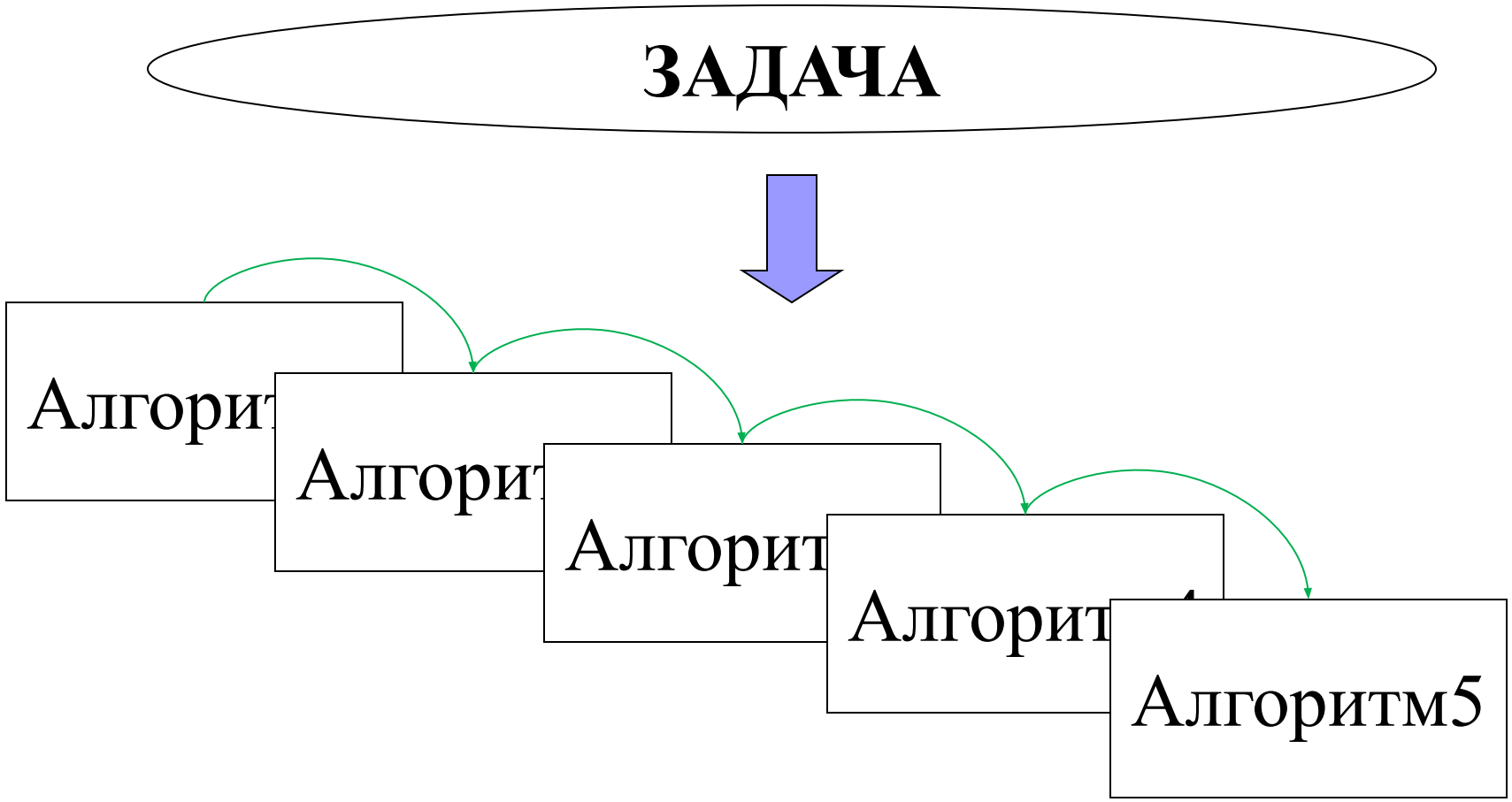
Основные парадигмы программирования

Процедурное программирование

Алгоритмическая декомпозиция



Значение функций

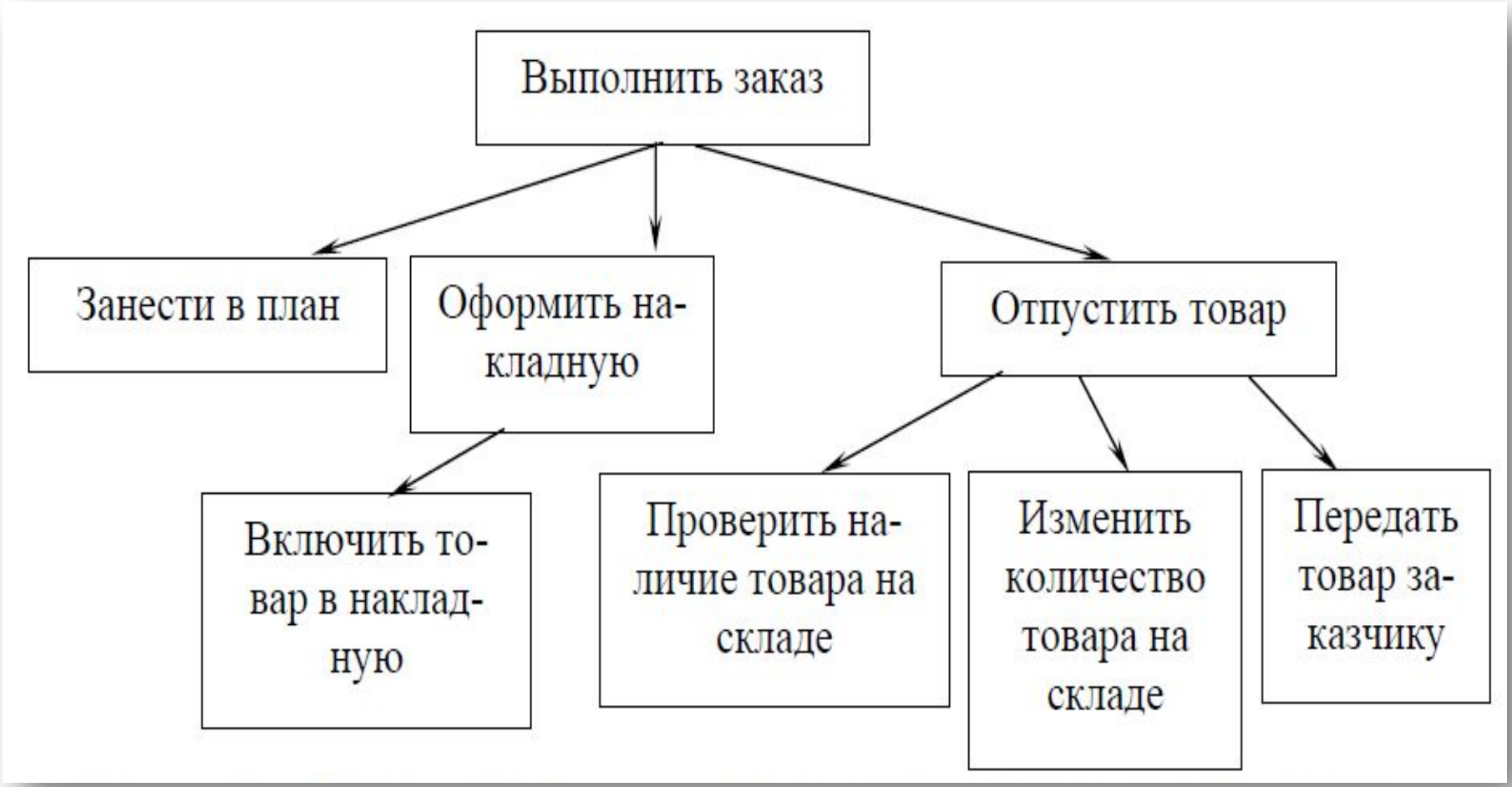
- Проблема – увеличение размера и сложности программ
- Решение – разделение сложных и больших программ на небольшие легко управляемые части, называемые *функциями:*
  - каждая функция в программе должна выполнять определенную задачу

Если программе необходимо выполнить какую-либо задачу, то она *вызывает* соответствующую функцию, обеспечивая эту функцию информацией, которая ей понадобится в процессе обработки.

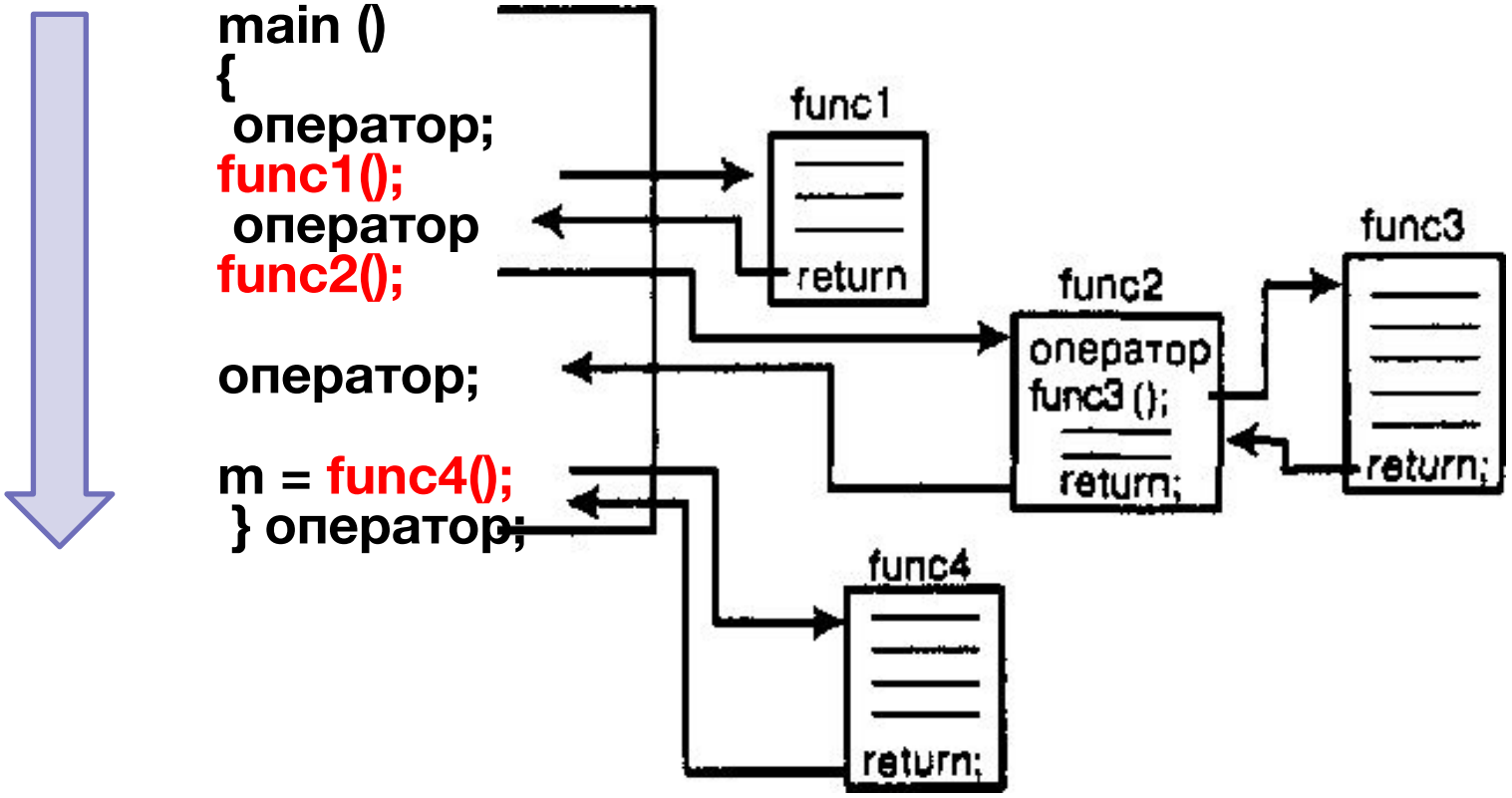
Виды функций

- Различают два вида функций:
  - определяемые пользователем (нестандартные)
  - встроенные (стандартные)
- Встроенные функции являются составной частью пакета компилятора и предоставляются фирмой-изготовителем
- Нестандартные функции создаются самим программистом

Алгоритмическая декомпозиция. Пример



Работа с функциями



- Когда программа вызывает функцию, управление переходит к телу функции, а затем выполнение программы возобновляется со строки, следующей после вызова

Описание функции в C++, C#

- Определение (описание) функции состоит из двух частей: заголовка и тела:

```
<тип> <имя функции> (<список параметров>)
{
        <тело функции>
}
```

  - **тип** - определяет тип значения, которое возвращает функция с помощью оператора return,
    по умолчанию функция возвращает значение (типа int).
  - **список параметров** - состоит из перечня типов и имен параметров, разделенных запятыми, круглые скобки обязательны.
  - **тело функции –** набор выражений.

Описание функции в Python

- Определение (описание) функции состоит из двух частей: заголовка и тела:

```
def <имя функции> (<список параметров>):
        <тело функции>
        return
```

  - **список параметров** - состоит из перечня типов и имен параметров, разделенных запятыми, круглые скобки обязательны.
  - **тело функции –** набор выражений.

Описание функции в Python

Пример def01.py

- Определение (описание) функции состоит из двух частей: заголовка и тела:

```
def <имя функции> (<список параметров>):
        <тело функции>
        return
```

Инструкция def действует как оператор присваивания

- создаются новый объект-функция и ссылка на объект с указанным именем, которая указывает на объект-функцию

Типы функций

Пример def00.py

- В Python можно создать четыре типа функций
  - глобальные функции
  - локальные функции
  - лямбда-функции
  - методы

Процедуры или Функции?

Процедура – вспомогательный алгоритм, который выполняет некоторые действия.

- текст (алгоритм) процедуры записывается **до** её вызова в основной программе
- чтобы процедура заработала, нужно **вызвать** её по имени из основной программы или из другой процедуры

Объявление процедуры:

define определить

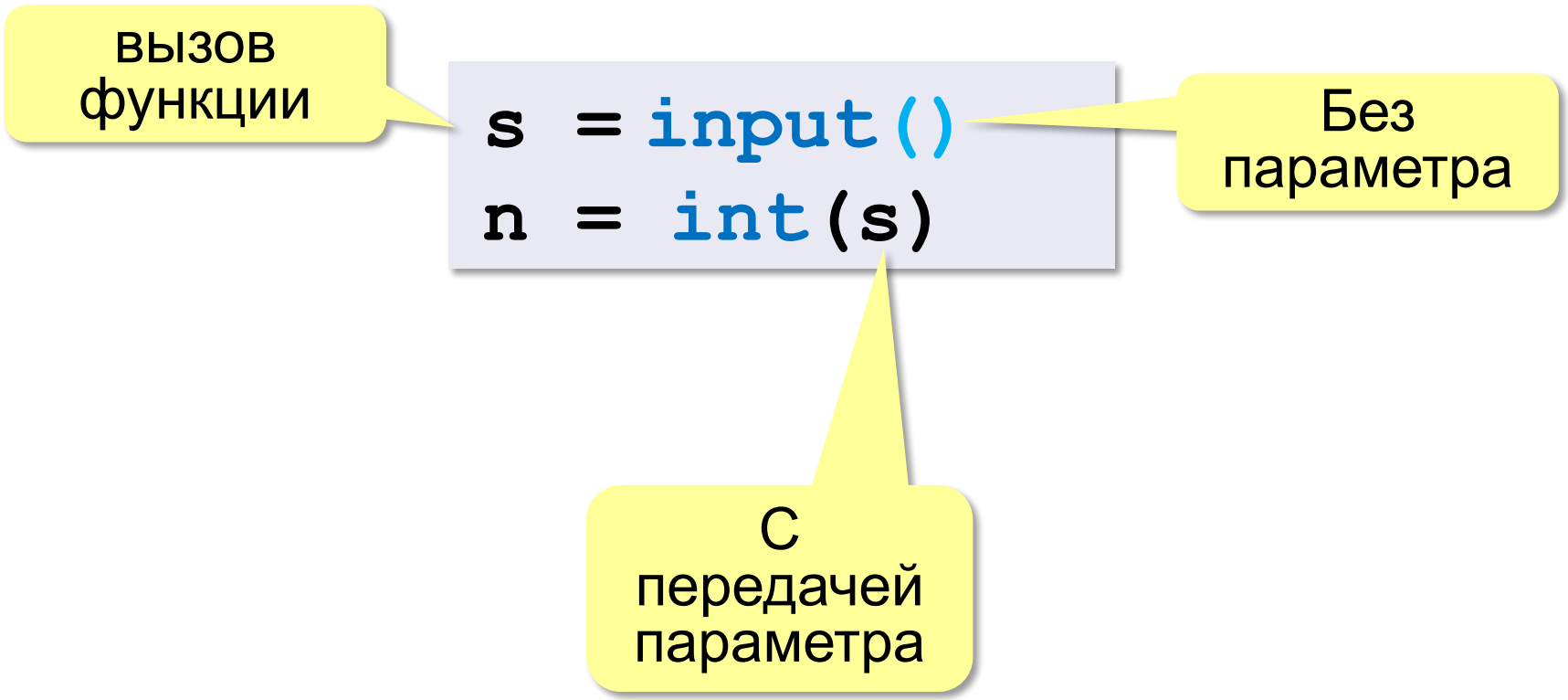
```
def ErrorMess():
  print("Ошибка в алгоритме")
```

вызов процедуры

```
n = int (input())
if n < 0:
   ErrorMess()
```

Процедуры или Функции?

Функция – это вспомогательный алгоритм, который возвращает *значение-результат* (число, символ или объект другого типа).



Использование параметров

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

Параметры – данные – вход в процедуру или функцию

```
def printBin( n ):
    k = 128
    while k > 0:
        print (n // k, end = "")
        n = n % k;
        k = k // 2
```

локальная переменная

```
printBin(178)
```

значение параметра (аргумент)

Несколько параметров:

```
def printSred(a, b):
    print ( (a + b)/2 )
```

Локальные и глобальные переменные

Пример def03.py

глобальная переменная

локальная переменная

```
a = 52
def qq():
    a = 1
    print (a)
qq()
print (a)
```

1

52

```
a = 5
def qq():
    print (a)
qq()
```

5

```
a = 5
def qq():
    global a
    a = 1
qq()
print (a)
```

работаем с глобальной переменной

1

Процедура и глобальные переменные

```
x = 5; y = 10


xSum()
```

```
def xSum():
    print ( x+y )
```

Так делать нельзя

1) процедура связана с глобальными переменными, нельзя перенести в другую программу
2) печатает сумму только **x** и **y**, нельзя напечатать сумму других переменных или сумму **x*y** и **3x**

Как исправить?

передавать данные через параметры

Процедура и глобальные переменные

Глобальные:

x	y
5	10

z	w
17	3

```
x = 5; y = 10
Sum2( x, y )
z=17; w=3
Sum2( z, w )
Sum2( z+x, y*w )
```

```
def Sum2(a, b):
    print( a+b )
```

Локальные:

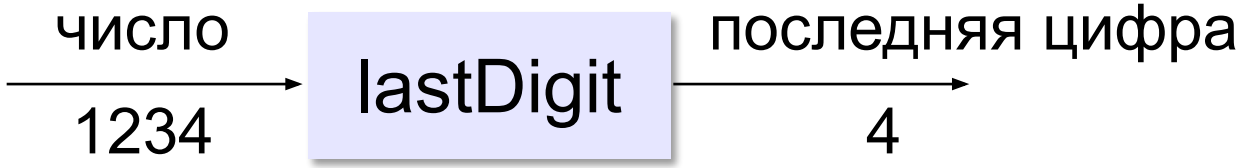
a	b
5	10
17	3
22	30

15
20
52

1) процедура не зависит от глобальных переменных
2) легко перенести в другую программу
3) печатает сумму любых выражений

Процедуры или Функции?

Пример. Создать функцию, которая вычисляет младшую цифру числа (разряд единиц).




```
def lastDigit( n ):
    d = n % 10
    return d
```

результат работы функции – значение **d**

передача результата

```
# вызов функции
k = lastDigit( 1234 )
print( k )
```

Процедуры или Функции?

Пример. Написать функцию, которая вычисляет сумму цифр числа.

```
def sumDigits( n ):
    sum = 0
    while n!= 0:
        sum += n % 10
        n = n // 10
    return sum

# основная программа
rez = sumDigits(12345)
print (rez)
```

Объявление функции

передача результата

передача аргумента

получение результата

Применение функций

```
x = 2*sumDigits( n+5 )
z = sumDigits( k ) + sumDigits( m )
if sumDigits( n ) % 2 == 0:
   print ( "Сумма цифр чётная" )
   print ( "Она равна", sumDigits( n ) )
```

Функция, возвращающая результат, может использоваться везде, где допускается такой тип

Одна функция вызывает другую:

```
def middle ( a, b, c ):
   mi = min ( a, b, c )
   ma = max ( a, b, c )
   return a + b + c - mi - ma
```

Вызываются функции min и max

Что вычисляет?

Полиморфность функций

Пример def02.py

В языке Python именно объекты определяют синтаксический смысл операции

Функции являются полиморфными - то есть они могут обрабатывать объекты произвольных типов, при условии, что они поддерживают ожидаемый интерфейс

```
def function1(first,second):
    c = first + second
    return c
```

Вызываются функции

```
resSum = function1(a,function1(a,b))
```

```
newL = function1(my_list, my_list)
```

Что вычисляет – если a b числа, a my_list список?

Проблема: как вернуть несколько значений?

Пример defReturn_many.py

Пример def_pack_unpack.py

```
def divmod( x, y ):
    d = x // y
    m = x % y
    return d, m
```

d – частное,
m – остаток

```
a, b = divmod( 7, 3 )
print( a, b )      # 2 1
```

```
q = divmod( 7, 3 )
print( q )         # (2, 1)
```

кортеж – набор элементов

- Объектом возврата нескольких значений может быть объект структуры, а также массив (C++, C#)

Проверка условий – логические функции

Логическая функция – это функция, возвращающая логическое значение (True/False).

```
def even(n):
    if n % 2 == 0:
        return True
    else:
        return False
```

```
def even(n):
    return (n % 2 == 0)
```

Применение логической функции

```
k = int(input())
if even( k ):
    print( "Число", k, "чётное" )
else:
    print( "Число", k, "нечётное" )
```

Необязательные параметры

- Объявление необязательных параметров разрешает пропуск их при указании аргументов

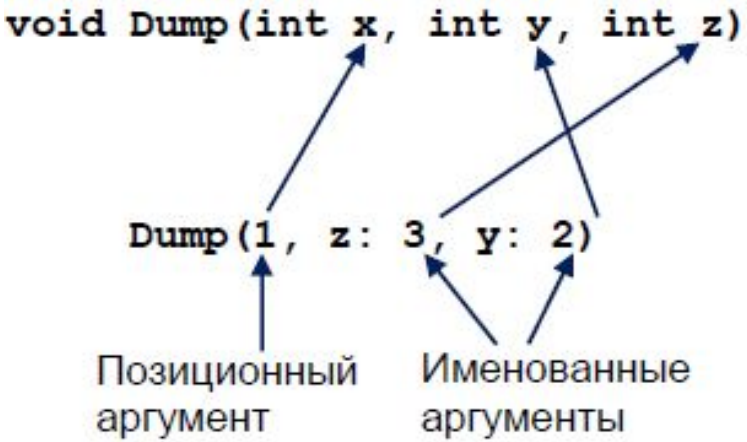

```
Dump(1, 2, 3);

Dump(1, 2);
Dump(1);
```

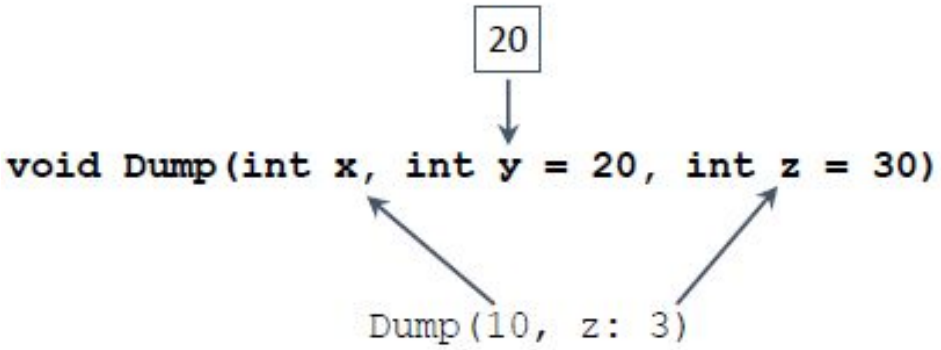
```
>>> def func(a, b, c=2): # c - необязательный аргумент
...     return a + b + c
...
>>> func(1, 2)  # a = 1, b = 2, c = 2 (по умолчанию)
5
>>> func(1, 2, 3)  # a = 1, b = 2, c = 3
6
>>> func(a=1, b=3)  # a = 1, b = 3, c = 2
6
```

Все необязательные параметры должны располагаться после обязательных параметров

Именованные аргументы

- Идея именованных аргументов заключается в том, что при передаче значения аргумента можно также указать имя параметра, для которого предназначено это значение.
- Компилятор проверяет, есть ли параметр с таким именем, и применяет для него заданное значение





- Все именованные аргументы должны располагаться после позиционных аргументов — произвольно переключаться между стилями нельзя

Передача параметров по значению

- Механизм, используемый по умолчанию:
  - Значение аргумента копируется в формальный параметр метода
  - Значение параметра может меняться внутри метода
  - Это не влияет на значение аргумента, используемого при вызове
  - Типы параметра и аргумента должны быть одинаковыми или совместимыми

Передача параметров по ссылке

- Что такое параметры, передаваемые по ссылке?
  - Ссылка на область памяти
- Использование параметров, передаваемых по ссылке
  - Типы параметра и аргумента должны совпадать
  - Изменения параметра отразятся на аргументе, используемый при вызове метода
  - При попытке передать ссылку на неинициализированный параметр, компилятор выдаст ошибку

Передача параметров в функцию

- Передача по значению
- Передача по ссылке

Пример defPassParameter.py

```
a = 3
x = [ 1, 2 ]
def f1(a):
    a = a + 1
    return a

def f2(a):
    a[0] = a[0] + 1
    return a
```

```
print (a)       # 3
print (x[0])        # 1

print(f1(a))        # 4
print(f2(x))        # [2, 2]

print (a)       # 3
print (x[0])        # 2
```

Выбор способа передачи параметров

Выбор способа передачи параметров при создании процедуры (функции):

- входные параметры нужно передавать по значению,
- выходные – по ссылке.

Рекомендации по использованию функций

- Функции должны быть компактными
- Функция должна выполнять только одну операцию
- Используйте содержательные имена