Процедуры

Предположим, что в нескольких местах программы требуется выводить на экран сообщение об ошибке: «*Ошибка программы*». Это можно сделать, например, так:

```
print ( "Ошибка программы" )
```

Конечно, можно вставить этот оператор вывода везде, где нужно вывести сообщение об ошибке. Но тут есть две сложности. Во-первых, строка-сообщение хранится в памяти много раз. Во-вторых, если мы задумаем поменять текст сообщения, нужно будет искать эти операторы вывода по всей программе. Для таких случаев в языках программирования предусмотрены *процедуры* – вспомогательные алгоритмы, которые выполняют некоторые действия.

```
def Error():
  print ( "Ошибка программы" )
n = int ( input() )
if n < 0:
  Error()
```

Процедура, выделенная белым фоном, начинается с ключевого слова **def** (от англ. *define* – определить). После имени процедуры ставятся пустые скобки (чуть далее мы увидим, что они могут быть и непустые!) и двоеточие. Тело процедуры записывается с отступом.

Фактически мы ввели в язык программирования новую команду **Error**, которая была расшифрована прямо в теле программы. Для того, чтобы процедура заработала, в основной программе (или в другой процедуре) необходимо ее *вызвать* по имени (не забыв скобки).

Процедура должна быть определена к моменту её вызова, то есть должна быть выполнена инструкция **def**, которая создает объект-процедуру в памяти. Если процедура вызывается из основной программы, то нужно поместить её определение раньше точки вызова.

Как мы видели, использование процедур сокращает код, если какие-то операции выполняются несколько раз в разных местах программы. Кроме того, большую программу разбивают на несколько процедур для удобства и упрощения, оформляя в виде процедур отдельные этапы сложного алгоритма. Такой подход делает всю программу более понятной.

Процедура с параметрами

Процедура **Error** при каждом вызове делает одно и то же. Более интересны процедуры, которым можно передавать *параметры* (или аргументы) – дополнительные данные, которые изменяют выполняемые действия.

Предположим, что в программе требуется многократно выводить на экран запись целого числа (0..255) в 8-битном двоичном коде. Старшая цифра в такой записи – это частное от деления числа на 128. Далее возьмем остаток от этого деления и разделим на 64 – получается вторая цифра и т.д. Алгоритм, решающий эту задачу для переменной **n**, можно записать так:

```
n = 125
k = 128
while k > 0:
  print ( n // k, end = "" )
  n = n % k
  k = k // 2
```

Обратим внимание на вызов функции **print**, у которой указан именованный параметр[10] **end** – завершающий символ (по умолчанию – символ «новая строка», который обозначается как «**\n**»). Напомним, что раньше мы уже использовали ещё один именованный параметр функции **print** –

---

[9] Эта задача известна как парадокс Монти Холла, потому что её решение противоречит интуиции и «здравому смыслу».

[10] Так называют параметры, имеющие имена.

sep – разделитель между элементами списка вывода (по умолчанию – пробел).

Писать такой цикл каждый раз, когда нужно вывести двоичное число, очень утомительно. Кроме того, легко сделать ошибку или опечатку, которую будет сложно найти. Поэтому лучше оформить этот вспомогательный алгоритм в виде процедуры. Но этой процедуре нужно передать *параметр* – число для перевода в двоичную систему. Программа получается такая:

```
def printBin(n):
  k = 128
  while k > 0:
    print ( n // k, end = "")
    n = n % k
    k = k // 2
# основная программа
printBin ( 99 )
```

Основная программа содержит всего одну команду – вызов процедуры **printBin**. В скобках указано значение параметра (его называют аргументом процедуры) – 99.

В заголовке процедуры в скобках записывают внутреннее имя параметра (то есть имя, по которому к нему можно обращаться в процедуре).

В процедуре используется *локальная* (внутренняя) переменная **k** – она известна только внутри этой процедуры, обратиться к ней из основной программы и из других процедур невозможно. Параметры процедуры – это тоже локальные переменные.

Если переменной присвоено значение в основной программе (вне всех процедур), она называется *глобальной*. Внутри процедуры можно обращаться к глобальным переменным, например, эта программа выведет значение 5:

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

Однако для того, чтобы изменить значение глобальной переменной (не создавая локальную), в процедуре с помощью слова **global** надо указать, что мы используем именно глобальную переменную. Следующая программа выведет на экран число 1:

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

Параметров может быть несколько, в этом случае они перечисляются в заголовке процедуры через запятую. Например, процедуру, которая выводит на экран среднее арифметическое двух чисел, можно записать так:

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