Пример функции

С функциями вы уже знакомы, потому что применяли встроенные функции языка Python (например, **input**, **int**, **randint**, см. § 56.). Функция, как и процедура – это вспомогательный алгоритм, который может принимать аргументы. Но, в отличие от процедуры, функция всегда возвращает *значение-результат*. Результатом может быть число, символ, символьная строка или любой другой объект.

Составим функцию, которая вычисляет сумму цифр числа. Будем использовать следующий алгоритм (предполагается, что число записано в переменной **n**):

```
sum = 0
while n!= 0:
  sum += n % 10
  n = n // 10
```

Чтобы получить последнюю цифру числа (которая добавляется к сумме), нужно взять остаток от деления числа на 10. Затем последняя цифра отсекается, и мы переходим к следующей цифре. Цикл продолжается до тех пор, пока значение **n** не становится равно нулю.

Пока остается неясно, как указать в программе, чему равно значение функции. Для этого используют оператор **return** (в переводе с англ. – «вернуть»), после которого записывают значение-результат:

```
def sumDigits( n ):
  sum = 0
  while n!= 0:
    sum += n % 10
    n = n // 10
  return sum
# основная программа
print ( sumDigits(12345) )
```

Так же как и в процедурах, в функциях можно использовать локальные переменные. Они входят в «зону видимости» только этой функции, для всех остальных функций и процедур они недоступны.

В функции может быть несколько операторов **return**, после выполнения любого из них работа функции заканчивается.

Функции, созданные таким образом, применяются точно так же, как и стандартные функции. Их можно вызывать везде, где может использоваться выражение того типа, который возвращает функция. Приведем несколько примеров:

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

Функцию можно вызывать не только из основной программы, но и из другой функции. Например, функция, которая находит среднее из трёх различных чисел (то есть число, заключённое между двумя остальными), может быть определена так:

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

Она использует встроенные функции **min** и **max**. Идея решения состоит в том, что если из суммы трёх чисел вычесть минимальное и максимальное, то получится как раз третье число.

Функция может возвращать несколько значений. Например, функцию, которая вычисляет сразу и частное, и остаток от деления двух чисел[11] можно написать так:

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

При вызове такой функции её результат можно записать в две различные переменные

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

Если указать только одну переменную, мы получим *кортеж* – набор элементов, который заключается в круглые скобки:

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

Логические функции

Часто применяют функции, которые возвращают *логическое значение* (**True** или **False**). Иначе говоря, такая *логическая* функция отвечает на вопрос «да или нет?» и возвращает 1 бит информации.

Вернёмся к задаче, которую мы уже рассматривали: вывести на экран все простые числа в диапазоне от 2 до 1000. Алгоритм определения простоты числа оформим в виде функции. При этом его можно легко вызвать из любой точки программы.

Запишем основную программу на псевдокоде:

```
for i in range(2,1001):
  if i - простое:
    print ( i )
```

Предположим, что у нас уже есть логическая функция **isPrime**, которая определяет простоту числа, переданного ей как параметр, и возвращает истинное значение (**True**), если число простое, и ложное (**False**) в противном случае. Такую функцию можно использовать вместо выделенного блока алгоритма:

```
if isPrime ( i ):
  print ( i )
```

Остаётся написать саму функцию **isPrime**. Будем использовать уже известный алгоритм: если число **n** в интервале от 2 до $\sqrt{n}$ не имеет ни одного делителя, то оно простое[12]:

```
def isPrime ( n ):
  k = 2
  while k*k <= n and n % k != 0:
    k += 1
  return (k*k > n)
```

Эта функция возвращает *логическое* значение, которое определяется как

```
k*k > n
```

Если это условие истинно, то функция возвращает **True**, иначе – **False**.

Логические функции можно использовать так же, как и любые условия: в условных операторах и циклах с условием. Например, такой цикл останавливается на первом введённом составном числе:

```
n = int ( input() )
while isPrime(n):
  print ( n, "- простое число" )
  n = int ( input() )
```