В большинстве языков программирования, в том числе и в Python, реализовано достаточное количество стандартных функций, например функции abs(x) и sqrt(x).

Кроме встроенных и библиотечных функций есть возможность писать собственные функции, предварительно описав их. Это даёт возможность выносить фрагменты кода, которые часто повторяются, в функции, что сокращает объём программы и делает её более понятной. Сначала рассмотрим пример использования стандартных функций:

x = int(input())

**print**(abs(x))

В этом примере используются четыре функции: input, int, abs и print.

Функции могут принимать какие-то значения в качестве параметров, но могут и не принимать никаких параметров. Например, функция input не получает входных значений.

Также функции могут возвращать какие-то значения, но могут ничего не возвращать. Например, функция print ничего не возвращает.

Описание функции в Python располагается в любом месте программы, но до первого её использования. После описания функции её можно использовать в выражениях наряду со стандартными функциями.

При описании функции указывается служебное слово **def**, затем после пробела имя функции, после чего в круглых скобках через запятую перечисляются её параметры. Если функция не принимает входных параметров, то в круглых скобках ничего не пишут. После круглых скобок ставится двоеточие, и затем на следующей строке с отступом задаётся тело функции так, как это делается при написании циклов.

Если функция должна вернуть какое-то значение, то для этого используется служебное слово **return**. Инструкция **return** завершает работу функции и возвращает значение соответствующей переменной (выражения). Инструкция **return** может встречаться в произвольном месте функции, её исполнение завершает работу функции и возвращает указанное значение в место вызова. Если функция не возвращает значения, то инструкция **return** используется без возвращаемого значения или инструкция **return** может отсутствовать.

Например, функция нахождения максимума из двух чисел будет выглядеть следующим образом:

**def** max(a, b):

**if** a > b:

**return** a

**else**:

**return** b

После того как мы написали такую функцию, мы можем ее использовать, например, вот так:

**print**(max(3, 8))

В результате эта строчка напечатает число 88.

Мы можем использовать уже написанные функции для реализации других функций. Например, мы можем реализовать функцию max3, находящую максимум трёх чисел, следующим образом:

**def** max3(a, b, c):

**return** max(max(a, b), c)

Функция max3 дважды вызывает функцию max для двух чисел: сначала чтобы найти максимум из a и b, потом чтобы найти максимум из этой величины и c. В программе вызов функции max3 может выглядеть так:

x = -2

y = 5

**print**(max3(2, -3 \* x, y))

или так:

a = int(input())

b = int(input())

m = max3(172, a, b - c)

Для примера напишем функцию min\_divisor, которая для заданного натурального числа n>1n>1 находит его минимальный делитель, больший единицы:

**def** min\_divisor(n):

**for** d **in** range(2, n + 1):

**if** n % d == 0:

**return** d

Эту же функцию можно реализовать с использованием цикла **while** следующим образом:

**def** min\_divisor(n):

d = 2

**while** n % d != 0:

d += 1

**return** d

На основании этой функции можно написать функцию is\_prime, которая проверяет число на простоту:

**def** is\_prime(n):

**return** n == min\_divisor(n)

Если число простое, то его минимальный делитель равен самому числу, и в таком случае функция is\_prime вернёт True. Если же число составное, то функция вернёт False.

Функцию is\_prime можно использовать, например, вот таким образом:

n = int(input())

**if** is\_prime(n):

**print**("Простое")

**else**:

**print**("Составное")