В описании процедуры или функции задаётся список формальных параметров. Каждый параметр, описанный в этом списке, является локальным по отношению к описываемой функции, то есть на него можно ссылаться по его имени только из данной функции, но не из основной программы или другой функции. Свое начальное значение он получает из соответствующего фактического параметра при вызове функции. Изменения, которые претерпевает формальный параметр, не влияют на значение фактического параметра. Соответствующее фактическое значение параметра должно иметь тип, совместимый с типом формального параметра. В общем случае можно сказать, что областью действия переменной является блок, в котором она описана.

Также в функции можно использовать глобальные переменные (объявленные вне функции, но доступные внутри функции) при условии, что имена этих глобальных переменных не совпадают с именами локальных переменных в функции.

Рассмотрим ряд примеров.

**def** f():

**print**(a)

a = 1

f()

Здесь переменной a присваивается значение 1 вне функции. Переменная a используется в функции f и является глобальной переменной. В момент вызова функции f глобальной переменной a присвоено значение 1, поэтому функция f выведет на экран число 1.

Если инициализировать какую-то переменную внутри функции, то использовать эту переменную вне функции не удастся. Например:

**def** f():

a = 1

f()

**print**(a)

При запуске этой программы получим ошибку во время исполнения:

NameError: name 'a' is not defined.

В этом примере переменная a объявлена внутри функции и является локальной переменной. При выходе из функции f локальная переменная a стала недоступной для использования, поэтому мы получили ошибку.

Интересным получится результат, если попробовать изменить значение глобальной переменной внутри функции:

**def** f():

a = 0

**print**(a)

a = 1

f()

**print**(a)

Будут выведены числа 0 и 1. То есть несмотря на то, что значение переменной a изменилось внутри функции, вне функции оно осталось прежним. Это сделано в целях “защиты” глобальных переменных от случайного изменения их внутри функции. Если внутри функции модифицируется значение некоторой переменной, то переменная с таким именем становится локальной переменной, и её модификация не приведёт к изменению глобальной переменной с таким же именем.

Рассмотрим такой пример:

**def** f():

**print**(a)

a = 0

**print**(a)

a = 1

f()

Может показаться, что эта программа сначала напечатает число 1, а потом напечатает число 0. Но это не так. Дело в том, что интерпретатор Python считает переменную локальной, если внутри функции есть хотя бы одна инструкция, модифицирующая значение этой переменной (это может быть оператор =, += и так далее, или использование этой переменной в качестве параметра цикла **for**). Поэтому такая переменная должна быть инициализирована внутри функции до её первого использования.

Поэтому в примере выше мы получим ошибку во время исполнения:

UnboundLocalError: local variable 'a' referenced before assignment

При этом даже если инструкция, модифицирующая переменную, никогда не будет выполнена, интерпретатор это проверить не сможет, и переменная всё равно будет считаться локальной, например:

**def** f():

**print**(a)

**if** False:

a = 0

a = 1

f()

При запуске этой программы возникает такая же ошибка, как в предыдущем примере:

UnboundLocalError: local variable 'a' referenced before assignment

В функции f есть инструкция a = 0, поэтому переменная a становится локальной переменной, но так как эта переменная не инициализирована перед первым ее использованием, то мы получаем ошибку.

Чтобы функция могла изменить значение глобальной переменной, необходимо объявить эту переменную внутри функции как глобальную при помощи ключевого слова **global**:

**def** f():

**global** a

a = 0

**print**(a)

a = 1

f()

**print**(a)

В этом примере на экран будут выведены числа 0 и 0, так как переменная a объявлена как глобальная, и её изменение внутри функции приводит к тому, что и вне функции переменная a будет равна новому значению 0.