**Функции**

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")

**Локальные и глобальные переменные**

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 11 вне функции. Переменная a используется в функции f и является глобальной переменной. В момент вызова функции f глобальной переменной a присвоено значение 11, поэтому функция f выведет на экран число 11.

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 00 и 11. 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 11, а потом напечатает число 00. 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 00 и 00, 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 00.

**Символы. Таблица ASCII**

Рассмотрим вопрос сравнения строк в компьютере. Например, строка 'hello' меньше строки 'world', потому что буква hh идёт в алфавите раньше, чем буква ww.

Строки сравниваются в алфавитном (лексикографическом) порядке. Сначала сравниваются первые буквы, если они равны, то сравниваются вторые буквы и так далее.

Если при таком сравнении мы дошли до конца одной из строк, то получается что эта строка является префиксом другой строки. В таком случае более короткая строка считается меньшей.

Выпишем для примера несколько коротких строк в порядке их возрастания: aa, aaaa, abab, acac, bb, bbbb, bcbc.

Но что делать, если нужно сравнить строчную букву aa и прописную букву AA? Также может понадобиться сравнивать буквы с цифрами или другими символами. Все символы можно сравнивать между собой, потому что каждому символу приписан код (некоторый номер), и сравнение символов производится по этому коду. Код буквы aa меньше, чем код буквы bb, причём коды букв aa, bb, cc, dd, …… идут подряд по возрастанию.

Чтобы понять, как сравниваются между собой буквы различного регистра, а также другие символы, посмотрим на таблицу ASCII (American standard code for information interchange).

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 32 | (space) | 48 | 0 | 64 | @ | 80 | P | 96 | ` | 112 | p |
| 33 | ! | 49 | 1 | 65 | A | 81 | Q | 97 | a | 113 | q |
| 34 | " | 50 | 2 | 66 | B | 82 | R | 98 | b | 114 | r |
| 35 | # | 51 | 3 | 67 | C | 83 | S | 99 | c | 115 | s |
| 36 | $ | 52 | 4 | 68 | D | 84 | T | 100 | d | 116 | t |
| 37 | % | 53 | 5 | 69 | E | 85 | U | 101 | e | 117 | u |
| 38 | & | 54 | 6 | 70 | F | 86 | V | 102 | f | 118 | v |
| 39 | ' | 55 | 7 | 71 | G | 87 | W | 103 | g | 119 | w |
| 40 | ( | 56 | 8 | 72 | H | 88 | X | 104 | h | 120 | x |
| 41 | ) | 57 | 9 | 73 | I | 89 | Y | 105 | i | 121 | y |
| 42 | \* | 58 | : | 74 | J | 90 | Z | 106 | j | 122 | z |
| 43 | + | 59 | ; | 75 | K | 91 | [ | 107 | k | 123 | { |
| 44 | , | 60 | < | 76 | L | 92 | \ | 108 | l | 124 | | |
| 45 | - | 61 | = | 77 | M | 93 | ] | 109 | m | 125 | } |
| 46 | . | 62 | > | 78 | N | 94 | ^ | 110 | n | 126 | ~ |
| 47 | / | 63 | ? | 79 | O | 95 | \_ | 111 | o | 127 | (del) |

В этой таблице приведены коды всех основных символов, включая все буквы латинского алфавита, цифры и знаки препинания.

Символы с кодами от 00 до 3131 являются специальными символами, и большинство из них в наше время не используются, поэтому эту часть таблицы мы не стали иллюстрировать.

Символ с кодом 3232 — это пробел. После него идут знаки препинания и знаки арифметических операций.

Символы с кодами 4848 — 5959 — это цифры от 00 до 99.

Прописные буквы латинского алфавита пронумерованы подряд кодами 6565 — 9090.

Строчные буквы латинского алфавита пронумерованы подряд кодами 9797 — 122122.

Для того чтобы можно было использовать символы других алфавитов, используется стандарт Unicode. В него включено более 100000100000 различных символов, среди которых можно встретить все распространённые национальные алфавиты мира. Этот стандарт позволяет сравнивать между собой не только буквы латинского (английского) алфавита, но и буквы различных алфавитов.

Для того чтобы узнать код символа, можно использовать функцию ord(). Например, строка кода

**print**(ord('A'))

напечатает число 6565 — код символа AA.

Функция ord() в качестве аргумента принимает строку, состоящую ровно из одного символа. Если передать ей пустую строку или строку длины больше 11, то получим ошибку исполнения.

Функция, которая по числу возвращает символ с данным кодом называется chr(). Например, код

**print**(chr(65))

напечатает символ AA, код которого равен 6565.

Давайте покажем, как по букве получить следующую за ней букву алфавита. Пусть нам даётся на вход не последняя буква алфавита. Считаем её и выведем на экран следующую за ней букву:

c = input()

**print**(chr(ord(c) + 1))

**Работа с символами**

На практике часто встречается задача о проверке пароля на надёжность. Пароль можно считать достаточно надёжным, если он не слишком короткий, и в нём встречаются хотя бы по одному разу цифра, строчная буква и заглавная буква. Давайте разберём, как такого рода проверки делать в языке Python.

Рассмотрим строку s. Пусть мы должны проверить её ii‍-‍й символ, то есть s[i]. Поместим его в переменную с. Чтобы проверить, является ли символ цифрой, нужно проверить, что его код находится между кодами цифр 00 и 99:

**if** ord('0') <= ord(c) <= ord('9'):

 **print**('Цифра')

Можно сравнивать не коды символов, а сами символы. Тогда получим следующий код:

**if** '0' <= c <= '9':

 **print**('Цифра')

Проверка, является ли символ прописной буквой латинского алфавита, будет выглядеть похожим образом:

**if** 'A' <= c <= 'Z':

И проверка, является ли символ строчной буквой латинского алфавита выглядит так:

**if** 'a' <= c <= 'z':

В языке Python есть встроенные методы для подобных проверок:

**if** c.isdigit(): *# проверка, является ли символ цифрой*

**if** c.isupper(): *# проверка, является ли символ прописной буквой*

**if** c.islower(): *# проверка, является ли символ строчной буквой*

**if** c.isalpha(): *# проверка, является ли символ буквой (строчной или прописной)*

Эти методы работают не только со строками длины 11, но и со строками произвольной длины. Метод будет возвращать True, если все символы строки удовлетворяют условию, которое этот метод проверяет. Эти методы работают не только с латинским алфавитом, но и со всеми другими алфавитами, которые включены в стандарт Unicode.

Предположим, что нам нужно перевести все буквы в заданном тексте в заглавный или в строчный регистр. Научимся это делать. Пусть нам подают на вход один символ, являющийся прописной буквой латинского алфавита. Требуется превратить этот символ в строчный и вывести на экран. Это можно сделать следующим образом:

c = input()

c = chr(ord(c) + 32)

**print**(c)

Тут мы воспользовались тем свойством, что код прописной и строчной буквы латинского алфавита всегда отличается ровно на 3232. Если нужно превратить букву из строчной в заглавную, то необходимо написать такой же код, но только число 3232 нужно вычитать, а не прибавлять.

В языке Python есть стандартные методы upper и lower для решения задачи о переводе символа в другой регистр. Например, код, написанный выше, можно реализовать следующим образом:

c = input()

c = c.upper()

**print**(c)

Методы upper и lower работают со строками любой длины и со всеми алфавитами.