A function is a piece of code in a program. The function performs a specific task. The advantages of using functions are:

* Reducing duplication of code
* Decomposing complex problems into simpler pieces
* Improving clarity of the code
* Reuse of code
* Information hiding

Functions in Python are first-class citizens. It means that functions have equal status with other objects in Python. Functions can be assigned to variables, stored in collections, or passed as arguments. This brings additional flexibility to the language.

There are two basic types of functions. Built-in functions and user defined ones. The built-in functions are part of the Python language. Examples are: dir(), len(), or abs(). The user defined functions are functions created with the def keyword.

**Defining functions**

A function is created with the def keyword. The statements in the block of the function must be indented.

def function():

print “Hello world”

The def keyword is followed by the function name with round brackets and a colon. The indented statements form a *body* of the function.

The function is later executed when needed. We say that we *call* the function. If we call a function, the statements inside the function body are executed. They are not executed until the function is called.

myfunc()

To call a function, we specify the function name with the round brackets.

**The return keyword**

A function is created to do a specific task. Often there is a result from such a task. The returnkeyword is used to return values from a function. A function may or may not return a value. If a function does not have a return keyword, it will send None.

returning.py

#!/usr/bin/python

# returning.py

def showMessage(msg):

print msg

def cube(x):

return x \* x \* x

x = cube(3)

print x

showMessage("Computation finished.")

print showMessage("Ready.")

We have two functions defined. One uses the return keyword, the other one does not.

def showMessage(msg):

print msg

The showMessage() function does not return explicitly a value. It shows a message on the console.

def cube(x):

return x \* x \* x

The cube() functions computes an expression and returns its result with the return keyword.

x = cube(3)

In this line we call the cube() function. The result of the computation of the cube() function is returned and assigned to the x variable. It holds the result value now.

# def Statements with Parameters

When you call the print() or len() function, you pass in values, called arguments in this context, by typing them between the parentheses. You can also define your own functions that accept arguments. Type this example into the file editor and save it as helloFunc2.py:

❶ def hello(name):

❷ print('Hello ' + name)

❸ hello('Alice')

hello('Bob')

When you run this program, the output looks like this:

Hello Alice

Hello Bob

The definition of the hello() function in this program has a parameter called name ❶. A parameter is a variable that an argument is stored in when a function is called. The first time the hello() function is called, it’s with the argument 'Alice' ❸. The program execution enters the function, and the variable name is automatically set to 'Alice', which is what gets printed by the print() statement ❷.

One special thing to note about parameters is that the value stored in a parameter is forgotten when the function returns. For example, if you added print(name) after hello('Bob') in the previous program, the program would give you a NameError because there is no variable named name. This variable was destroyed after the function call hello('Bob') had returned, so print(name) would refer to a name variable that does not exist.

This is similar to how a program’s variables are forgotten when the program terminates. I’ll talk more about why that happens later in the chapter, when I discuss what a function’s local scope is.

Type casting…

Below is a table of the conversion functions in Python and their examples.

|  |  |  |
| --- | --- | --- |
| **Function** | **Converting what to what** | **Example** |
| int() | string, floating point → integer | >>> int('2014')  2014  >>> int(3.141592)  3 |
| float() | string, integer → floating point number | >>> float('1.99')  1.99  >>> float(5)  5.0 |
| str() | integer, float, list, tuple, dictionary → string | >>> str(3.141592)  '3.141592'  >>> str([1,2,3,4])  '[1, 2, 3, 4]' |
| list() | string, tuple, dictionary → list | >>> list('Mary') # list of characters in 'Mary'  ['M', 'a', 'r', 'y']  >>> list((1,2,3,4)) # (1,2,3,4) is a tuple  [1, 2, 3, 4] |
| tuple() | string, list → tuple | >>> tuple('Mary')  ('M', 'a', 'r', 'y')  >>> tuple([1,2,3,4]) # [ ] for list, ( ) for tuple  (1, 2, 3, 4) |