A function packages a computation consisting of multiple steps into a form that can be easily understood and reused. (The person in the image to the left is in the middle of executing the function "make two cups of espresso".) In this chapter, you will learn how to design and implement your own functions. Using the process of stepwise refinement, you will be able to break up complex tasks into sets of cooperating functions.

5.1 Functions as Black Boxes

A function is a named sequence of instructions.

A **function** is a sequence of instructions with a name. You have already encountered several functions. For example, the `round` function, which was introduced in Chapter 2, contains instructions to round a floating-point value to a specified number of decimal places.

You *call* a function in order to execute its instructions. For example, consider the following program statement:

```
price = round(6.8275, 2)   # Sets result to 6.83
```

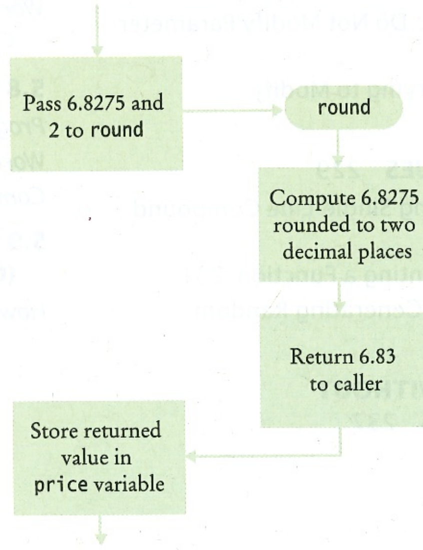
By using the expression `round(6.8275, 2)`, your program *calls* the `round` function, asking it to round 6.8275 to two decimal digits. The instructions of the `round` function execute and compute the result. The `round` function *returns* its result back to where the function was called and your program resumes execution (see Figure 1).

Arguments are supplied when a function is called.

When another function calls the `round` function, it provides "inputs", such as the values `6.8275` and `2` in the call `round(6.8275, 2)`. These values are called the **arguments** of the function call. Note that they are not necessarily inputs provided by a human user. They are simply the values for which we want the function to compute a result. The "output" that the `round` function computes is called the **return value**.

The return value is the result that the function computes.

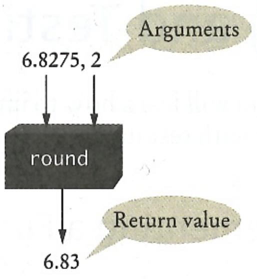
Functions can receive multiple arguments, but they return only one value. It is also possible to have functions with no arguments. An example is the `random` function that requires no argument to produce a random number.



Pass 6.8275 and 2 to round
round
Compute 6.8275 rounded to two decimal places
Return 6.83 to caller
Store returned value in price variable

Figure 1
Execution Flow of a Function Call

Figure 2
The round Function as a Black Box



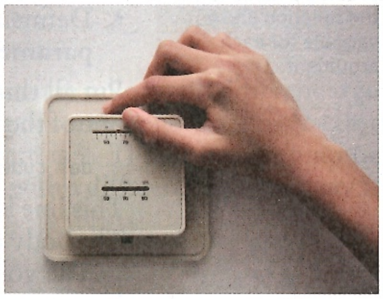
The return value of a function is returned to the point in your program where the function was called. It is then processed according to the statement containing the function call. For example, suppose your program contains a statement

```
price = round(6.8275, 2)
```

When the `round` function returns its result, the return value is stored in the variable `price`.

Do not confuse returning a value with producing program output. If you want the return value to be printed, you need to add a statement such as `print(price)`.

At this point, you may wonder how the `round` function performs its job. For example, how does `round` compute that 6.8275 rounded to two decimal digits is 6.83? Fortunately, as a user of the function, you *don't need to know* how the function is implemented. You just need to know the *specification* of the function: If you provide arguments *x* and *n*, the function returns *x* rounded to *n* decimal digits. Engineers use the term *black box* for a device with a given specification but unknown implementation. You can think of `round` as a black box, as shown in Figure 2.



Although a thermostat is usually white, you can think of it as a "black box". The input is the desired temperature, and the output is a signal to the heater or air conditioner.

When you design your own functions, you will want to make them appear as black boxes to other programmers. Those programmers want to use your functions without knowing what goes on inside. Even if you are the only person working on a program, making each function into a black box pays off: there are fewer details that you need to keep in mind.

SELF CHECK

1. Consider the function call `round(3.14159, 2)`. What are the arguments and return values?
2. What is the return value of the function call `round(round(4.499, 2), 0)`?
3. The `ceil` function in the `math` module of the Python standard library is described as follows: The function receives a single numerical argument *a* and returns the smallest `float` value $\geq a$ that is an integer. What is the return value of `ceil(2.3)`?
4. It is possible to determine the answer to Self Check 3 without knowing how the `ceil` function is implemented. Use an engineering term to describe this aspect of the `ceil` function.

Practice It Now you can try these exercises at the end of the chapter: R5.3, R5.5.