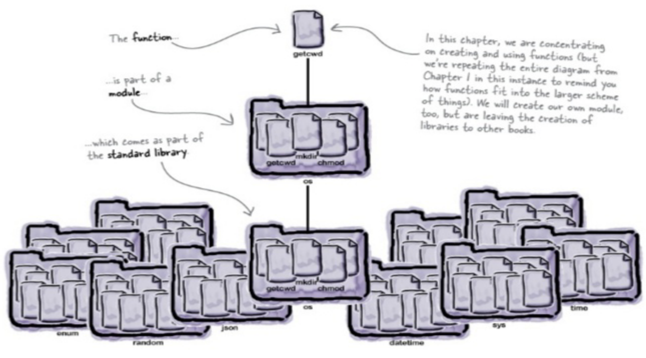
Reusing code is key to building a maintainable system.

And when it comes to reusing code in Python, it all starts and ends with the humble function. Take some lines of code, give them a name, and you've got a function (which can be reused). Take a collection of functions and package them as a file, and you've got a module (which can also be reused). It's true what they say: it's good to share, and by the end of this chapter, you'll be well on your way to sharing and reusing your code, thanks to an understanding of how Python's functions and modules work.

Reusing Code with Functions

Although a few lines of code can accomplish a lot in Python, sooner or later you're going to find your program's codebase is growing...and, when it does, things quickly become harder to manage. What started out as 20 lines of Python code has somehow ballooned to 500 lines or more! When this happens, it's time to start thinking about what strategies you can use to reduce the complexity of your codebase.

Like many other programming languages, Python supports modularity, in that you can break large chunks of code into MIsmaller, more manageable pieces. You do this by creating functions, which you can think of as named chunks of code. Recall this diagram from Chapter 1, which shows the relationship between functions, modules, and the standard library:


The **function**...
getcwd
...is part of a **module**...
getcwd
mkdir
chmod
os
...which comes as part of the **standard library**
In this chapter, we are concentrating on creating and using functions (but we're repeating the entire diagram from Chapter 1 in this instance to remind you how functions fit into the larger scheme of things). We will create our own module, too, but are leaving the creation of libraries to other books.
enum
random
json
os
datetime
sys
time

In this chapter, we're going to concentrate on what's involved in creating your own functions, shown at the very top of the diagram. Once you're happily creating functions, we'll also show you how to create a module.

Introducing Functions

Before we get to turning some of our existing code into a function, let’s spend a moment looking at the anatomy of any function in Python. Once this introduction is complete, we’ll look at some of our existing code and go through the steps required to turn it into a function that you can reuse.

Don’t sweat the details just yet. All you need to do here is get a feel for what functions look like in Python, as described on this and the next page. We’ll delve into the details of all you need to know as this chapter progresses. The IDLE window on this page presents a template you can use when creating any function. As you are looking at it, consider the following:

Functions introduce two new keywords: def and return

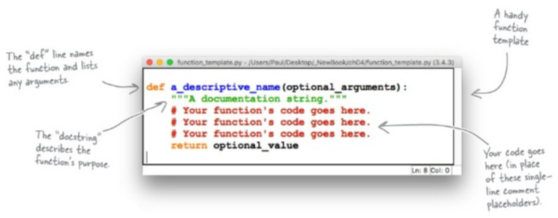
Both of these keywords are colored orange in IDLE. The def keyword names the function (shown in blue), and details any arguments the function may have. The use of the return keyword is optional, and is used to pass back a value to the code that invoked the function.

Functions can accept argument data

A function can accept argument data (i.e., input to the function). You can specify a list of arguments between the parentheses on the def line, following the function’s name.

Functions contain code and (usually) documentation

Code is indented one level beneath the def line, and should include comments where it makes sense. We demonstrate two ways to add comments to code: using a triple-quoted string (shown in green in the template and known as a docstring), and using a single-line comment, which is prefixed by the # symbol (and shown in red, below).


function_template.py - /Users/Paul/Desktop/_NewBook/ch04/function_template.py (3.4.3)
Ln: 8 Col: 0
A handy function template
The "def" line names the function and lists any arguments.
The "docstring" describes the function's purpose.
Your code goes here (in place of these single-line comment placeholders).

```
def a_descriptive_name(optional_arguments):
    """A documentation string."""
    # Your function's code goes here.
    # Your function's code goes here.
    # Your function's code goes here.
    return optional_value
```