Programming Tip 5.1

Function Comments

Whenever you write a function, you should *comment* its behavior. Comments are for human readers, not compilers. Various individuals prefer different layouts for function comments. In this book, we will use the following layout:

```
## Computes the volume of a cube.
#  @param sideLength the length of a side of the cube
#  @return the volume of the cube
#
def cubeVolume(sideLength) :
   volume = sideLength ** 3
   return volume
```

Function comments explain the purpose of the function, the meaning of the parameter variables and return value, as well as any special requirements.

This particular documentation style is borrowed from the Java programming language. It is supported by a wide variety of documentation tools such as Doxygen (`www.doxygen.org`), which extracts the documentation in HTML format from the Python source.

Each line of the function comment begins with a hash symbol (#) in the first column. The first line, which is indicated by two hash symbols, describes the purpose of the function. Each `@param` clause describes a parameter variable and the `@return` clause describes the return value.

There is an alternative (but, in our opinion, somewhat less descriptive) way of documenting the purpose of a Python function. Add a string, called a "docstring", as the first statement of the function body, like this:

```
def cubeVolume(sideLength) :
   "Computes the volume of a cube."
   volume = sideLength ** 3
   return volume
```

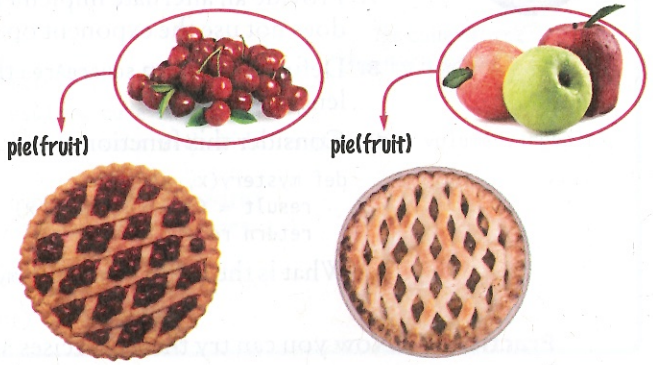
We don't use this style, but many Python programmers do.

Note that the function comment does not document the implementation (*how* the function does what it does) but rather the design (*what* the function does, its inputs, and its results). The comment allows other programmers to use the function as a "black box".

5.3 Parameter Passing

Parameter variables hold the arguments supplied in the function call.

In this section, we examine the mechanism of parameter passing more closely. When a function is called, variables are created for receiving the function's arguments. These variables are called **parameter variables**. (Another commonly used term is **formal parameters**.) The values that are supplied to the function when it is called are the **arguments** of the call. (These values are also commonly called the **actual parameters**.) Each parameter variable is initialized with the corresponding argument.



A recipe for a fruit pie may say to use any kind of fruit. Here, "fruit" is an example of a parameter variable. Apples and cherries are examples of arguments.

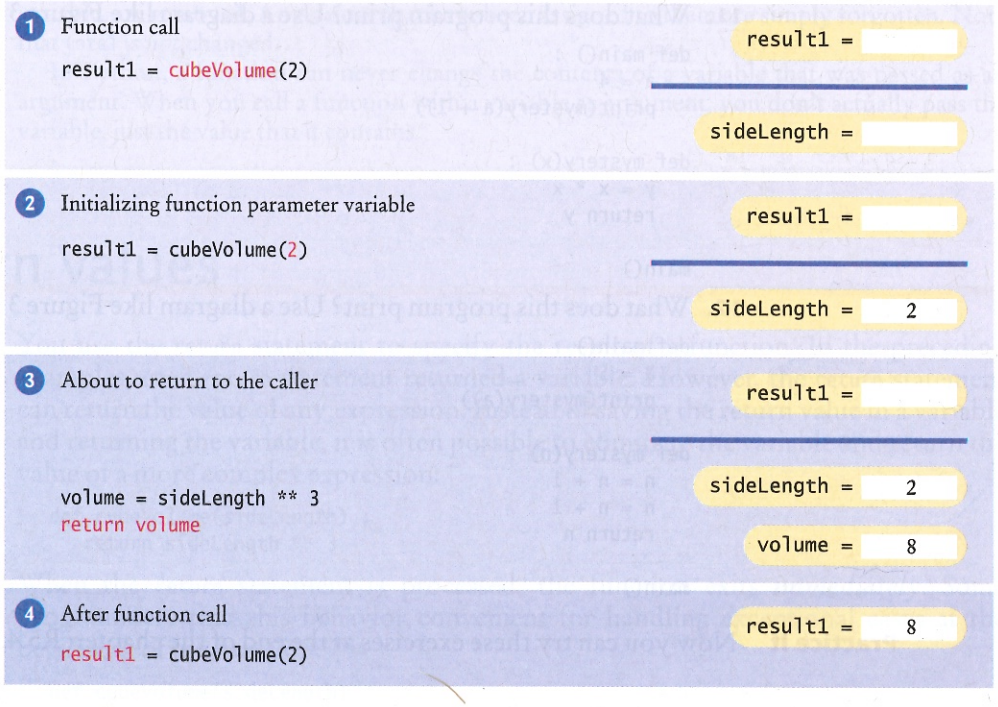


Figure 3
Parameter Passing

Consider the function call illustrated in Figure 3:

```
result1 = cubeVolume(2)
```

- The parameter variable `sideLength` of the `cubeVolume` function is created when the function is called. 1
- The parameter variable is initialized with the value of the argument that was passed in the call. In our case, `sideLength` is set to 2. 2
- The function computes the expression `sideLength ** 3`, which has the value 8. That value is stored in the variable `volume`. 3
- The function returns. All of its variables are removed. The return value is transferred to the *caller*, that is, the function calling the `cubeVolume` function. The caller puts the return value in the `result1` variable. 4

Now consider what happens in a subsequent call, `cubeVolume(10)`. A new parameter variable is created. (Recall that the previous parameter variable was removed when the first call to `cubeVolume` returned.) It is initialized with 10, and the process repeats. After the second function call is complete, its variables are again removed.

10. What does this program print? Use a diagram like Figure 3 to find the answer.

```
def main() :
   a = 5
   b = 7
   print(mystery(a, b))

def mystery(x, y) :
   z = x + y
   z = z / 2.0
   return z

main()
```

11. What does this program print? Use a diagram like Figure 3 to find the answer.

```
def main() :
   a = 4
   print(mystery(a + 1))

def mystery(x) :
   y = x * x
   return y

main()
```

12. What does this program print? Use a diagram like Figure 3 to find the answer.

```
def main() :
   a = 5
   print(mystery(a))

def mystery(n) :
   n = n + 1
   n = n + 1
   return n

main()
```

Practice It Now you can try these exercises at the end of the chapter: R5.4, R5.12, P5.8.

Programming Tip 5.2

Do Not Modify Parameter Variables

In Python, a parameter variable is just like any other variable. You can modify the values of the parameter variables in the body of a function. For example,

```
def totalCents(dollars, cents) :
   cents = dollars * 100 + cents    # Modifies parameter variable.
   return cents
```

However, many programmers find this practice confusing (see Common Error 5.1). To avoid the confusion, simply introduce a separate variable:

```
def totalCents(dollars, cents) :
   result = dollars * 100 + cents
   return result
```

Common Error 5.1

Trying to Modify Arguments

The following function contains a common error: trying to modify an argument.

```
def addTax(price, rate) :
   tax = price * rate / 100
   price = price + tax    # Has no effect outside the function.
   return tax
```

Now consider this call:

```
total = 10
addTax(total, 7.5)    # Does not modify total.
```

When the `addTax` function is called, `price` is set to the value of `total`, that is, 10. Then `price` is changed to 10.75. When the function returns, all of its variables, including the `price` parameter

variable, are removed. Any values that have been assigned to them are simply forgotten. Note that `total` is *not* changed.

In Python, a function can never change the contents of a variable that was passed as an argument. When you call a function with a variable as argument, you don't actually pass the variable, just the value that it contains.

5.4 Return Values

The return statement terminates a function call and yields the function result.

You use the `return` statement to specify the result of a function. In the preceding examples, each `return` statement returned a variable. However, the `return` statement can return the value of any expression. Instead of saving the return value in a variable and returning the variable, it is often possible to eliminate the variable and return the value of a more complex expression:

```
def cubeVolume(sideLength) :
   return sideLength ** 3
```

When the `return` statement is processed, the function exits *immediately*. Some programmers find this behavior convenient for handling exceptional cases at the beginning of the function:

```
def cubeVolume(sideLength)
   if sideLength < 0 :
      return 0
   # Handle the regular case.
   . . .
```

If the function is called with a negative value for `sideLength`, then the function returns 0 and the remainder of the function is not executed. (See Figure 4.)

Every branch of a function should return a value. Consider the following incorrect function:

```
def cubeVolume(sideLength) :
   if sideLength >= 0 :
      return sideLength ** 3
   # Error—no return value if sideLength < 0
```

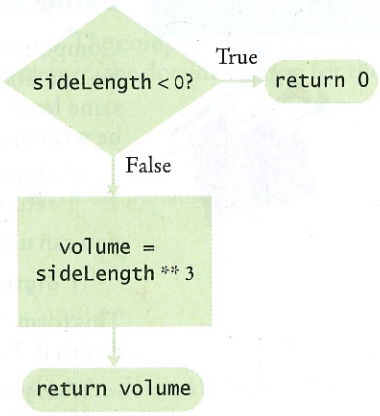


Figure 4 A `return` Statement Exits a Function Immediately