03 Prepare: Writing Functions

Because most useful computer programs are very large, programmers divide their programs into parts. Dividing a program into parts makes it easier to write, debug, and understand the program. A programmer can divide a Python program into modules, classes, and functions. In this lesson and the next, you will learn how to write your own functions.

Videos

Watch the following four videos from Microsoft about writing functions:

[Introducing Functions](#) (10 minutes)

[Demonstration: Functions](#) (8 minutes)

[Parameterized Functions](#) (7 minutes)

[Demonstration: Parameterized Functions](#) (5 minutes)

Concepts

Here are the Python programming concepts and topics that you should learn during this lesson.

What Is a Function?

A function is a group of statements that together perform one task. Broadly speaking, there are four types of functions in Python which are:

1. Built-in functions
2. Standard library functions
3. Third-party functions
4. User-defined functions

In the previous lesson, you learned how to call the first two types of functions. In lesson 5, you will learn how to install third-party modules and call third-party functions. In this lesson, you will learn how to write and call user-defined functions.

What Is a User-Defined Function?

A user-defined function is a function that is not a built-in function, a standard function, or a third-party function. A user-defined function is written by a programmer like yourself as part of a program. For some students the term "user-defined function" is confusing because the user of a program doesn't define the function. Instead, the programmer (you) define user-defined functions. Perhaps a more correct term is programmer-defined function. Writing user-defined functions has several advantages, including:

1. making your code more reusable
2. making your code easier to understand and debug
3. making your code easier to change and add capabilities

How to Write a User-Defined Function

To write a user-defined function in Python, simply type code that matches this template:

```
def function_name(param1, param2, … paramN):
    """documentation string"""
    statement1
    statement2
        ⋮
    statementN
    return value
```

The first line of a function is called the header or signature, and it includes the following:

1. the keyword `def` (which is an abbreviation for "define")
2. the function name
3. the parameter list (with the parameters separated by commas)

Here is the header for a function named `draw_circle` that takes three parameters named `x`, `y`, and `radius`:

```
def draw_circle(x, y, radius):
```

You could read the previous line of code as, "Define a function named `draw_circle` that takes three parameters named *x*, *y*, and *radius*."

The function name must start with a letter or the underscore (_). The rest of the name must be made of letters, digits (0–9), or the underscore. A function name cannot include spaces or other punctuation. A function name should be meaningful and should describe briefly what the function does. Well-named functions often start with a verb.

The statements inside a function are called the body of the function. Just like other block statements, such as `if`, `else`, `while`, and `for`, all of which end with a colon (:), you must indent the statements inside the body of a function. The body of a function should begin with a documentation string which is a triple quoted string that describes the function's purpose, parameters and return value. The body of a function may contain as many statements as you wish to write inside of it. However, it is a good idea to limit functions to less than 20 lines of code.

Example 1 contains a function named `print_cylinder_volume()` with no parameters that gets two numbers from the user: *radius* and *height* and uses those numbers to compute the volume of a cylinder and then prints the volume for the user to see.

```python
# Example 1

import math

# Define a function named print_cylinder_volume.
def print_cylinder_volume():
    """Compute and print the volume of a cylinder.
    Parameters: none
    Return: nothing
    """
    # Get the radius and height from the user.
    radius = float(input("Enter the radius of a cylinder: "))
    height = float(input("Enter the height of a cylinder: "))

    # Compute the volume of the cylinder.
    volume = math.pi * radius**2 * height

    # Print the volume of the cylinder.
    print(f"Volume: {volume:.2f}")
```

Because the `print_cylinder_volume` function in example 1 doesn't accept parameters, it must be called without any arguments like this:

```
print_cylinder_volume()
```

How to Make a User-Defined Function Reusable

Because the `print_cylinder_volume` function in example 1 gets input from a user and prints its results to a terminal window, it can be used only in a program that runs when a user is present. It cannot be used in a program that runs automatically and gets input from a file or the network or a sensor. In other words, the `print_cylinder_volume` function in example 1 is not reusable in other programs. The most reusable functions are ones that take parameters, perform calculations, and return a value but *do not perform user input and output*.

The parameter list in a function's header contains data stored in variables that the function needs to complete its task. A parameter is a variable whose value comes from outside the function. One way to get input into a function is to ask the user for input by calling the built-in Python `input` function. Another way to get input into a function is through the function's parameters. Getting input through parameters is much more flexible than asking the user for input because the input through parameters can come from the user or a file on a hard drive or the network or a sensor or even another function.

Example 2 contains another version of the `print_cylinder_volume` function. This second version doesn't get the radius and height from the user. Instead, it gets input through its two parameters named *radius* and *height*.

```python
# Example 2

import math

# Define a function named print_cylinder_volume.
def print_cylinder_volume(radius, height):
    """Compute and print the volume of a cylinder.
    Parameters
        radius: the radius of the cylinder
        height: the height of the cylinder
    Return: nothing
    """
    # Compute the volume of the cylinder.
    volume = math.pi * radius**2 * height

    # Print the volume of the cylinder.
    print(volume)
```

Because the second version of `print_cylinder_volume` function accepts two parameters, it must be called with two arguments like this:

```python
print_cylinder_volume(2.5, 4.1)
```

To return a result from a function, simply type the keyword `return` followed by whatever result you want returned to the calling function. Example 3 contains a third version of the cylinder volume function. Notice that the version in example 3 returns the volume instead of printing it, which makes the function more reusable. Notice also in example 3 that we changed the name of the function from `print_cylinder_volume` to `compute_cylinder_volume` because this version doesn't print the volume but instead returns it.

```python
# Example 3

import math

# Define a function named computer_cylinder_volume.
def compute_cylinder_volume(radius, height):
    """Compute and return the volume of a cylinder.
    Parameters
        radius: the radius of the cylinder
```

```
        height: the height of the cylinder
    Return: the volume of the cylinder
    """
    # Compute the volume of the cylinder.
    volume = math.pi * radius**2 * height

    # Return the volume of the cylinder so that the
    # volume can be used somewhere else in the program.
    return volume
```

Many functions that you've used in the past such as `input`, `float`, and `round`, return a result. When a function returns a result, we usually write code to store that returned result in a variable to use later in the program like this:

```
text = input("Please enter your name: ")
```

Because the `compute_cylinder_volume` function in example 3 accepts two parameters and returns a value, it could be called like this:

```
volume = compute_cylinder_volume(2.5, 4.1)
```

The User-Defined Function Named `main`

In all previous Python programs that you wrote in CSE 110 and 111, you wrote statements that were not in a function like the simple program in example 4.

```
1   # Example 4
2
3   import math
4
5   # Get the radius and height from the user.
6   radius = float(input("Enter the radius of a cylinder: "))
7   height = float(input("Enter the height of a cylinder: "))
8
9   # Compute the volume of the cylinder.
10  volume = math.pi * radius**2 * height
11
12  # Print the volume of the cylinder.
13  print(f"Volume: {volume:.2f}")
```

```
> python example_4.py
Enter the radius in centimeters: 3
Enter the height in centimeters: 8
Volume: 226.19
```

Writing statements outside a function can lead to poor organization within a large program. Professional software developers write statements inside a function whenever possible. Beginning with this lesson, we will write nearly all statements inside a user-defined function. Also, each program will have a user-defined function named `main` which will contain the beginning statements of the program. In addition, each program will have one or more user-defined functions that perform calculations and other useful work and return a value to the call point. Example 5 contains the same Python program as example 4 except most of the statements are inside a user-defined function named `main`.

```
1   # Example 5
2
3   import math
4
5   # Define a function named main.
6   def main():
7       # Get the radius and height from the user.
8       radius = float(input("Enter the radius of a cylinder: "))
9       height = float(input("Enter the height of a cylinder: "))
10
11      # Compute the volume of the cylinder.
```

```
12         volume = math.pi * radius**2 * height
13
14         # Print the volume of the cylinder.
15         print(f"Volume: {volume:.2f}")
16
17  # Start this program by
18  # calling the main function.
19  main()
```

```
> python example_5.py
Enter the radius in centimeters:  3
Enter the height in centimeters:  8
Volume: 226.19
```

Notice the call to the `main` function at line 19 in example 5. Without that call to the `main` function, when we run the program, the program will not do anything. All of your future programs in CSE 111 will have a user-defined function named `main` and will have a call to `main` at the bottom of the program.

A Complete Program with a Reusable User-Defined Function

If you look closely at the code in examples 1 and 5, you will realize that both programs have the same problem, namely both the `print_cylinder_volume` function in example 1 and the `main` function in example 5 are not reusable because both of them get input from a user and print to a terminal window. A better way to write the program in examples 1 and 5 is to separate the program into two functions, one named `main` and one named `compute_cylinder_volume` as shown in example 6.

Example 6 contains a complete program with two functions, the first named `compute_cylinder_volume` at line 6 and the second named `main` at line 24. At line 31, the `main` function calls the `compute_cylinder_volume` function. Notice that the `compute_cylinder_volume` function gets its input through parameters and returns a value which makes this function reusable in other programs, including programs that run without a user.

```
1   # Example 6
2
3   import math
4
5   # Define a function that accepts two parameters.
6   def compute_cylinder_volume(radius, height):
7       """Compute and print the volume of a cylinder.
8       Parameters
9           radius: the radius of the cylinder
10          height: the height of the cylinder
11      Return: the volume of the cylinder
12      """
13      # Compute the volume of the cylinder.
14      volume = math.pi * radius**2 * height
15
16      # Return the volume of the cylinder so that the
17      # volume can be used somewhere else in the program.
18      # The returned result will be available wherever
19      # this function was called.
20      return volume
21
22
23  # Define the main function.
24  def main():
25      # Create two variables to hold the radius and height.
26      radius = float(input("Enter the radius of a cylinder: "))
27      height = float(input("Enter the height of a cylinder: "))
28
29      # Call the compute_cylinder_volume function and store
30      # its return value in a variable to use later.
31      volume = compute_cylinder_volume(radius, height)
32
33      # Print the volume of the cylinder.
34      print(f"Volume: {volume:.2f}")
```

```
35
36
37   # Start this program by
38   # calling the main function.
39   main()
```

```
> python example_6.py
Enter the radius in centimeters: [3]
Enter the height in centimeters: [8]
Volume: 226.19
```

What Happens When the Computer Calls a Function?

Some students have trouble visualizing what happens when the computer calls (executes) a function. The following code example is a complete program that includes several function calls. The green arrows in the example code show how the computer executes statements in one location and then jumps to execute statements in another function. The blue arrows show how data flows from arguments into parameters and from a returned result to a variable. The circled numbers show the order in which the events happen in the computer.
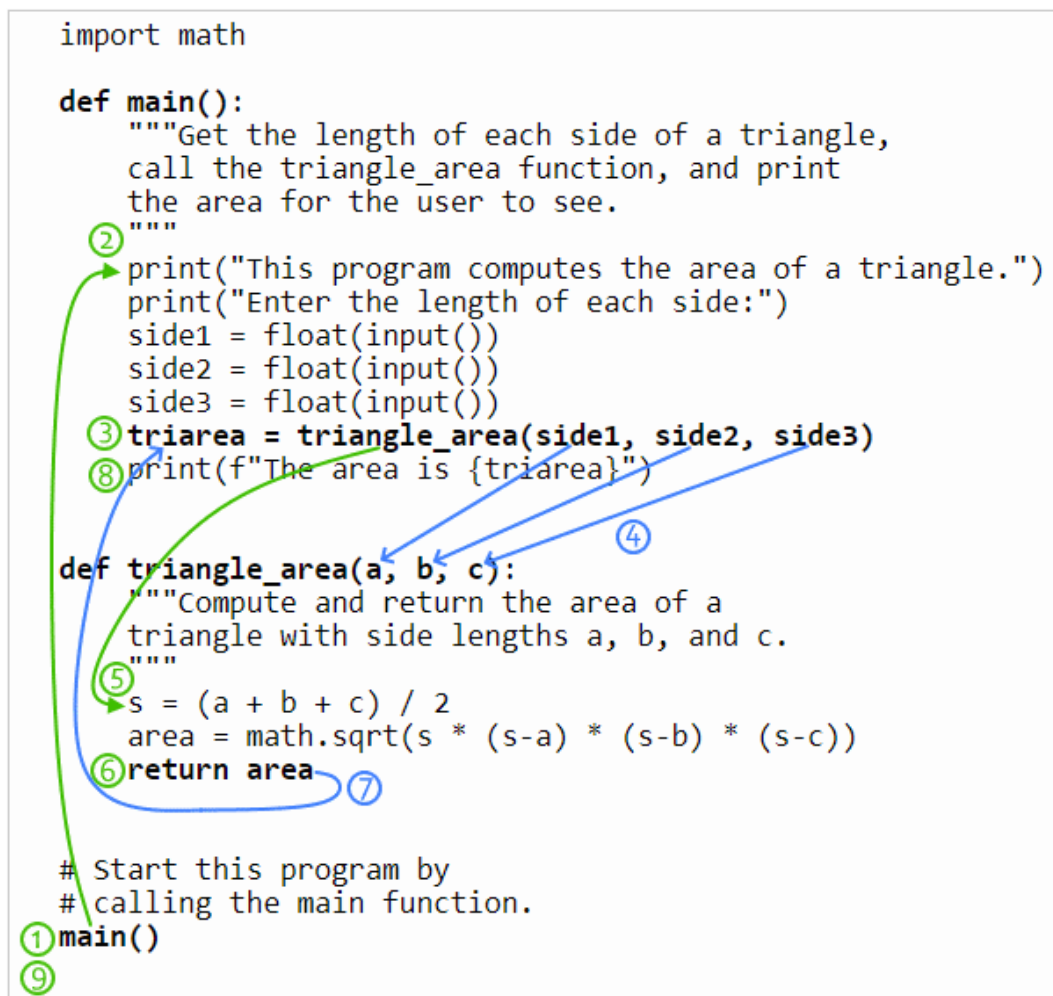
A computer will execute the statements in the previous example in the following order:

A. The statement at (1) is not inside a function, so the computer executes it when the program begins. The statement at (1) is a call to the **main** function which causes the computer to begin executing the statements inside **main** at (2).

B. At (2), the computer prints a description of the program and gets three numbers from the user.

C. The statement at (3) is a call to the **triangle_area** function which causes the computer to copy the values in the arguments *side1*, *side2*, and *side3* into the parameters *a*, *b*, and *c*, respectively and then begin executing the statements inside the **triangle_area** function at (5).

D. At (5), the computer computes the area of a triangle.

E. The statement at (6) is a return statement which causes the computer to stop executing the `triangle_area` function, to return the computed area to the call point at (3), and to resume executing statements at the call point.

F. At the call point (3), the computer stores the returned value in the *triarea* variable.

G. At (8), the computer prints the value that is in the *triarea* variable for the user to see. This is the last statement in the `main` function, so after executing it, the computer resumes executing the statements after the call point (1) to `main`.

H. At (9), there are no more statements after the call to `main`, so the computer terminates the program.

The most reusable functions are ones that take parameters, perform calculations, and return a value but *do not perform user input and output*. In the previous code example, there are two functions: `main` and `triangle_area`. The `main` function is certainly useful in this program, but it is not reusable in other programs because it gets user input and prints the result for the user to see. The `triangle_area` function is very reusable in another program because it doesn't get user input or print output. Instead, it takes three parameters, performs a calculation, and returns the result to the calling function. The `triangle_area` function is so reusable that it could be included in a library of functions that compute the area and volume of 2-D and 3-D geometric shapes.

Summary

A function is a group of statements that together perform one task. A user-defined function is a function written by a programmer like you. To write a user-defined function, write code that follows this template:

```
def function_name(param1, param2, … paramN):
    """documentation string"""
    statement1
    statement2
         ⋮
    statementN
    return value
```

To call a user-defined function, write code that follows this template:

```
variable_name = function_name(arg1, arg2, … argN)
```

The most reusable functions are ones that take parameters, perform calculations, and return a value but *do not perform user input and output*. All of your future programs in CSE 111 will have a user-defined function named `main` and will have a call to `main` at the bottom of the program.

It is extremely important that you can write and call functions. After watching the videos and reading the concepts related to writing a function, if the concepts still seem confusing or vague to you, watch the videos and read the list of concepts *again*.