BYU IDAHO CSE 111 | Programming with Functions

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 videos from Microsoft about writing functions:

- Introducing Functions (10 minutes)
- Demo: Functions (8 minutes)
- Parameterized Functions (7 minutes)
- Demo: Parameterized Functions (5 minutes)

Concepts

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

1. A function is a group of statements that together perform one task. Broadly speaking, there are four types of functions in Python which are:
   a. Built-in functions
   b. Standard library functions
   c. Third-party functions
   d. User-defined functions

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

2. 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 does not 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:
   a. making your code more reusable
   b. making your code easier to understand and debug
   c. mkaing your code easier to change and add capabilities

3. 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 1.

```
1  # Example 1
2
3  # Get a value in miles and convert it to kilometers.
4  miles = float(input("Please enter a distance in miles: "))
5  km = miles * 1.60934
6  print(f"{miles} miles is {km} kilometers")
```

   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 function. Also, each program will have a

function named `main` which will contain the beginning statements of the program. In addition, each program will have one or more functions that perform calculations and other useful work and return a value to the call point. Example 2 contains the same Python program as example 1 except most of the statements are inside a function named `main`.

```
 1  # Example 2
 2
 3  # Define a function named main.
 4  def main():
 5      # Get a value in miles and convert it to kilometers.
 6      miles = float(input("Please enter a distance in miles: "))
 7      km = miles * 1.60934
 8      print(f"{miles} miles is {km} kilometers")
 9
10  # Start this program by
11  # calling the main function.
12  main()
```

Notice the call to the `main` function at line 12 in example 2. Without that call to the `main` function, when we run the program, the program would not do anything. All of your future programs in CSE 111 will have a function named `main` and will have a call to `main` at the bottom of the program.

4. The `main` function in example 2 is 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:
   a. the keyword `def` (which is an abbreviation for "define")
   b. the function name
   c. 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 curtly what the function does. Well-named functions often start with a verb.

The parameter list 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.

5. 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 3 contains a function named `print_cylinder_volume()` 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.

```
# Example 3

import math

# Define a function named print_cylinder_volume.
def print_cylinder_volume():
    """Compute and print the volume of a cylinder."""

    # Get the radius and height from the user.
    radius = float(input("Please enter the radius of a cylinder: "))
    height = float(input("Please 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(volume)
```

Becuase the `print_cylinder_volume` function in example 3 doesn't accept parameters, it must be called without any arguments like this:

```
    print_cylinder_volume()
```

Example 4 contains another version of the `print_cylinder_volume` function. This second version doesn't get the radius and height from the user. Instead it accepts two parameters named *radius* and *height*.

```
# Example 4

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
    """
    # 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:

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

6. 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: ")
```

To return a result from a function, simply type the keyword `return` followed by whatever result you want returned to the calling function. Example 5 contains a third version of the cylinder volume function. Notice that this third version returns the volume instead of printing it, which makes the function more reusable. In this third version, 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.

```
# Example 5

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
```

Because the `compute_cylinder_volume` function in example 5 accepts two parameters and returns a value, it could be called like this:

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

Example 6 contains a complete program with two functions, the first named `compute_cylinder_volume` at line 6 and the seond named `main` at line 25. At line 32, the `main` function calls the `compute_cylinder_volume` function.

```
 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
 9      Parameters
10          radius: the radius of the cylinder
11          height: the height of the cylinder
12      Return: the volume of the cylinder
13      """
14      # Compute the volume of the cylinder.
15      volume = math.pi * radius**2 * height
16
17      # Return the volume of the cylinder so that the
```

```
18      # volume can be used somewhere else in the program.
19      # The returned result will be available wherever
20      # this function was called.
21      return volume
22
23
24  # Define the main function.
25  def main():
26      # Create two variables to hold the radius and height.
27      radius = float(input("Enter the radius in centimeters: "))
28      height = float(input("Enter the height in centimeters: "))
29
30      # Call the compute_cylinder_volume function and store
31      # its return value in a variable to use later.
32      volume = compute_cylinder_volume(radius, height)
33
34      if volume < 1000:
35          print(f"That cylinder holds {volume} cubic")
36          print("centimeters, which is less than a liter.")
37      elif volume > 1000:
38          print(f"That cylinder holds {volume} cubic")
39          print("centimeters, which is more than a liter.")
40      else:
41          print(f"That cylinder holds {volume} cubic")
42          print("centimeters, which is exactly one liter.")
43
44
45  # Start this program by
46  # calling the main function.
47  main()
```

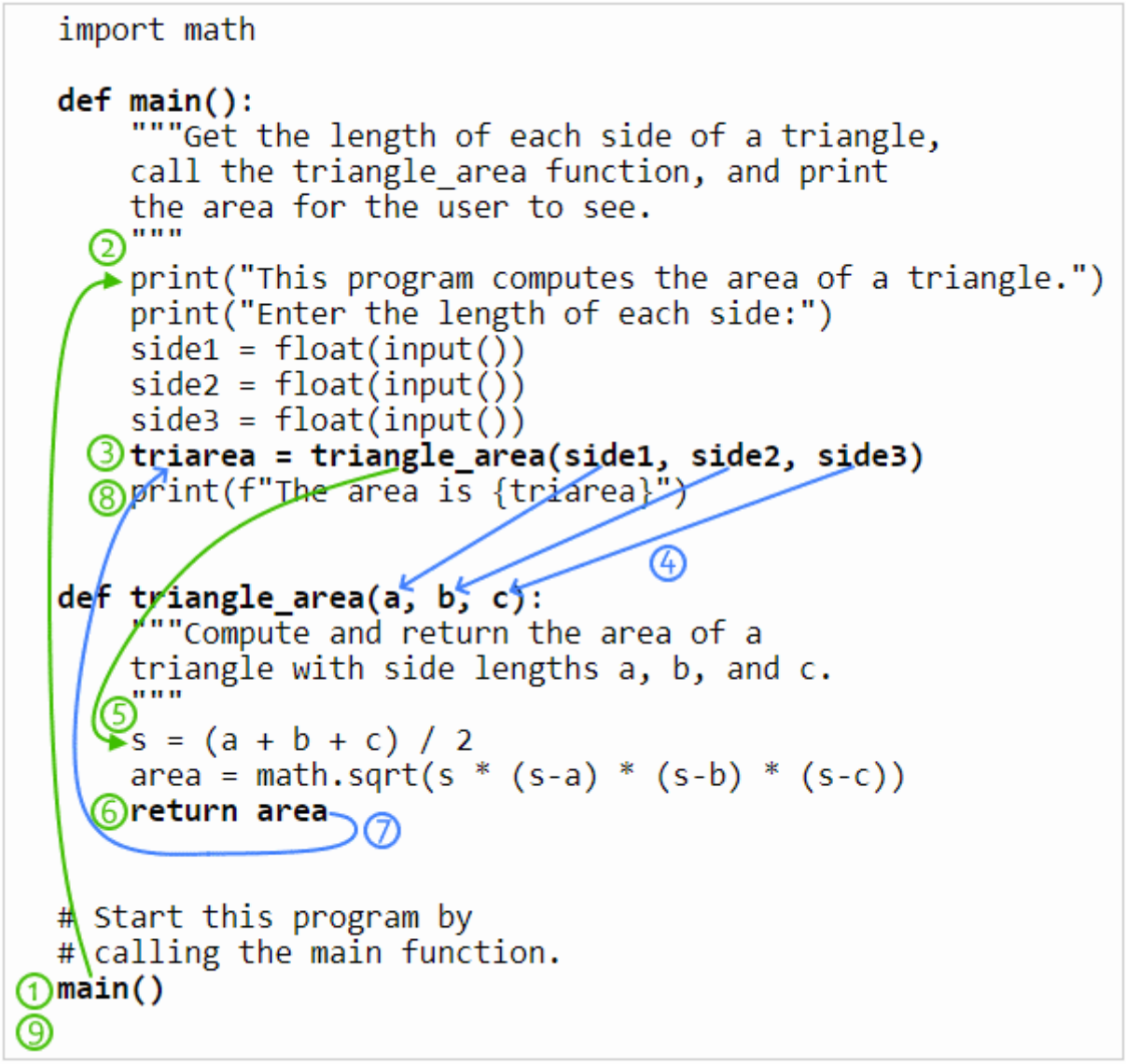
7. While it's usually a good practice, you don't *have* to store the return value of a function in a variable. Sometimes you'll see it used directly as shown in example 7 at lines 11 and 15.

```
 1  # Example 7
 2
 3  # Define the main function.
 4  def main():
 5      # Create two variables to hold the radius and height.
 6      radius = float(input("Enter the radius in centimeters: "))
 7      height = float(input("Enter the height in centimeters: "))
 8
 9      # Call the compute_cylinder_volume function
10      # and immediately print its return value.
11      print( compute_cylinder_volume(radius, height) )
12
13      # Call the compute_cylinder_volume function again
14      # and use its return value in an if statement.
15      if compute_cylinder_volume(radius, height) < 1000:
16          print("That cylinder holds less than a liter.")
17      elif compute_cylinder_volume(radius, height) > 1000:
18          print("That cylinder holds more than a liter.")
19      else:
20          print("That cylinder holds exactly one liter.")
21
22
23  # Start this program by
24  # calling the main function.
25  main()
```

Notice in example 7, there are three statements that call the `compute_cylinder_volume` function, one at line 11 to print the volume, another at line 15 to check if the volume is less than a liter, and yet another at line 17 to check if the volume is greater than a liter. Every time the computer calls a function, the computer will execute the code that is inside that function. In example 7, because the arguments are the same at lines 11, 15 and 17, the returned result will be the same in all three cases. So it would be faster to save the result in a variable and reuse the variable instead, as shown in example 8 at lines 11, 13, 15, and 17.

```
 1  #Example 8
 2
 3  # Define the main function.
 4  def main():
 5      # Create two variables to hold the radius and height.
 6      radius = float(input("Enter the radius in centimeters: "))
 7      height = float(input("Enter the height in centimeters: "))
 8
 9      # Call the compute_cylinder_volume function and store
10      # its return value in a variable to use later.
11      volume = compute_cylinder_volume(radius, height)
12
13      print(f"The volume is: {volume} cubic centimeters.")
14
15      if volume < 1000:
16          print("That cylinder holds less than a liter.")
17      elif volume > 1000:
18          print("That cylinder holds more than a liter.")
19      else:
20          print("That cylinder holds exactly one liter.")
21
22
23  # Start this program by
24  # calling the main function.
25  main()
```

8. 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.

9. 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.

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*.