Chapter 8 - User-Defined Functions

User-Defined Functions

Advanced Concepts

Learning Objectives - Function Composition

- **Define helper function**
- **Demonstrate two ways to implement helper functions**
- **Define function composition**
- **Identify two benefits to function composition**
- **Define function modularity**

Helper Functions

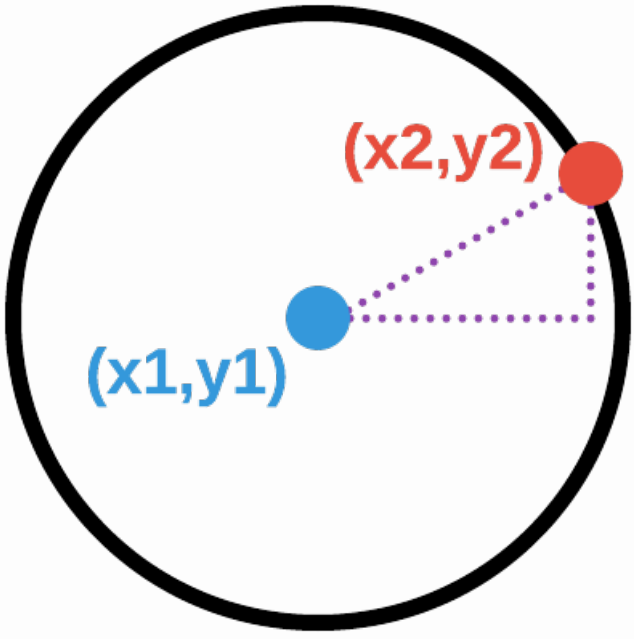
Helper Functions

Well written functions are those with a single, specific task. Complex problems will require more than one function for the solution. Helper functions are functions that are called from within other functions. Take, for example, the formula for calculating the area of a circle:

$$A = \pi r^2$$

It would be quite easy to write a Python function to calculate the area of a circle. However, instead of knowing the radius of the circle, you have the X/Y coordinates for a point at the center of the circle and another point on the circle. The distance formula (which is based on the Pythagorean Theorem) can calculate the radius of the circle.

$$\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$



Radius

The area function is dependent upon the distance formula. This is where helper functions come into play. Start by defining a function

radius. The square root function is included in the math module. Be sure to import math in your program. Then define a function area which calls radius. Since area requires radius, area also requires all of the parameters needed for the radius function. Finally, print the result of the area of a circle with the points (0, 0) and (4, 4).

```
import math

def radius(x1, y1, x2, y2):
    """Distance formula to determine the radius of a circle"""
    return(math.sqrt((x2 - x1)**2 + (y2 - y1)**2))

def area(x1, y1, x2, y2):
    """Area of a circle function"""
    return(math.pi * radius(x1, y1, x2, y2)**2)

print(area(0, 0, 4, 4))
```

challenge

Another way to do powers

Another way to do powers is to use math.pow. This function has two arguments, the base and the exponent. So 3 ** 2 becomes math.pow(3, 2). Rewrite the return statements for the two functions above using math.pow.

▶ **Solution**

Inner Functions

Python allows you to declare a function inside another function. Doing this hides the inner function from the main program. In the code below, only the area function can call the radius function.

```
import math

def area(x1, y1, x2, y2):
    """Area of a circle function"""
    def radius(x1, y1, x2, y2):
        """Distance formula to determine the radius of a circle"""
        return math.sqrt(math.pow(x2 - x1, 2) + math.pow(y2 - y1, 2))

    return(math.pi * math.pow(radius(x1, y1, x2, y2), 2))

print(area(0, 0, 4, 4))
```

challenge

What happens if you:

Try to call the `radius` function from your program?

```
print(radius(0, 0, 4, 4))
```

▶ **Why does this cause an error?**

Function Composition

Function Composition

Function composition is similar to helper functions in that two or more functions are used together to complete a larger task. However, function composition is a specific way in which functions are combined. Instead of calling the helper function from within another function, the helper function is called as a parameter of the function it is helping.

```
import math

def area(r):
    """Area of a circle"""
    return(math.pi * math.pow(r, 2))

def radius(x1, y1, x2, y2):
    """Distance formula to calculate the radius"""
    return(math.sqrt(math.pow(x2 - x1, 2) + math.pow(y2 - y1, 2)))

print(area(radius(0, 0, 4, 4)))
```

Function Composition

It is important to note that the function definition for `area` **does not** take a function as a parameter. It expects a value (the radius of the circle). There is an order of operation for function composition. The `radius` function is executed first, and then the value it returns is passed to the `area` function.

```
import math

def area(r):
    """Area of a circle"""
    return(math.pi * math.pow(r, 2))

def radius(x1, y1, x2, y2):
    """Distance formula to calculate the radius"""
    return(math.sqrt(math.pow(x2 - x1, 2) + math.pow(y2 - y1, 2)))

print(area(radius(0, 0, 4, 4)))
```

Function Composition vs Helper Functions

Both function composition and helper functions provide the same functionality. Using one over the other will not affect the end result. However, function composition offers improved readability. Look at the two function calls below:

```
circle_area1(x1, y1, x2, y2)
circle_area2(radius(x1, y1, x2, y2), math.pi)
```

Without seeing the function definitions, it is not clear how area can be derived from two points on a Cartesian plane in the first function call. The second function makes it very clear that the radius and pi are being used to calculate the area.

Another difference between the two is that helper functions can only be used with user-defined functions. You do not have access to the function definitions for the built-in functions for Python, so you cannot call a helper function from within a built-in function. You can, however, use function composition to combine built-in functions to accomplish a larger task.

```
import math

print(math.sqrt(int("25")))
```

challenge

What happens if you:

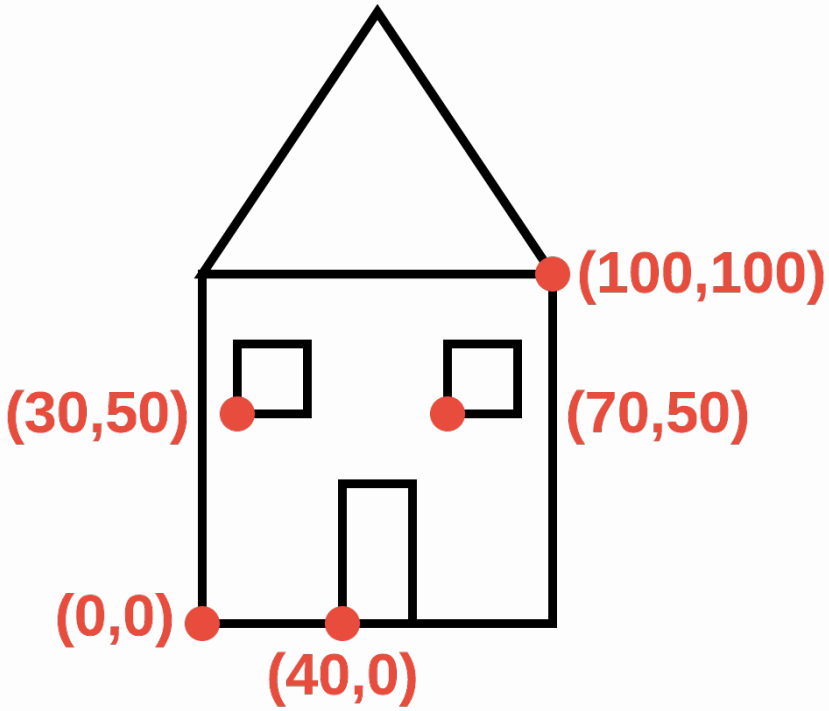
- Add the print statement `print(math.pow(int("2"), int("3")))`?

▶ **Limits of Function Composition and Readability**

Modularity

Modularity

Another benefit to using functions in your programs is modularity. Modularity means dividing a program into separate and independent units. In a previous lesson, you used Python's turtle graphics to draw a house. Think about how you would write functions to draw the following image. The red dots represents the starting points for each of the shapes. Look for repeating patterns and think about how these patterns could be represented in a modular way.



House

The image is composed of two shapes: a rectangle (a square is a rectangle with four sides of equal length) and a triangle. You also need the ability to reposition the turtle without drawing any lines on the screen. These are the three functions that will make up the drawing.

```
import turtle

t = turtle.Turtle()

def triangle(size):
    """Draw a triangle with a given size"""
    for i in range(3):
        t.lt(120)
        t.forward(size)

def rectangle(width, height):
    """Draw a rectangle with a given width and height"""
    for i in range(2):
        t.forward(width)
        t.lt(90)
        t.forward(height)
        t.lt(90)

def reposition(x, y):
    """Pick up the pen, move the turtle, set the
    direction of the turtle, and put the pen down"""
    t.penup()
    t.goto(x, y)
    t.setheading(0)
    t.pendown()

rectangle(100, 100) #draw the house
reposition(100, 100) #move to starting point for the roof
triangle(100) #draw the roof
reposition(40, 0) #move to starting point for the door
rectangle(20, 40) #draw the door
reposition(10, 50) #move to starting point for the left window
rectangle(20, 20) #draw the left window
reposition(70, 50) #move to starting point for the right window
rectangle(20, 20) #draw the right window

turtle.mainloop()
```

These functions are modular in that each one draws a shape. More importantly, the functions were written with parameters to specify the size of each shape. Modularity makes functions reusable. This way you

do not have write a function for the house, another for the door, and a third for the windows.

challenge

Tiny House

Change the code to draw a house that is half of the size of the original.

▶ **Solution**