04 Prepare: Function Details

During this lesson, you will learn additional details about writing and calling functions. These details include variable scope, default parameter values, and optional arguments and will help you understand functions better and use them more effectively.

Variable Scope

The scope of a variable determines how long that variable exists and where it can be used. Within a Python program, there are two categories of scope: local and global. A variable has local scope when it is defined (assigned a value) inside a function. A variable has global scope when it is defined outside of all functions. Here is a small Python program that has two variables: *g* and *x*. *g* is defined outside of all functions and therefore has global scope. *x* is defined inside the `main` function and therefore has local scope.

```
# g is a global variable because it
# is defined outside of all functions.
g = 25

def main():
    # x is a local variable because
    # it is defined inside a function.
    x = 1
```

As shown in the following table, a local variable (a variable with local scope) is defined inside a function, exists for as long as its containing function is executing, and can be used within its containing function but nowhere else. Parameters have local scope because they are defined within a function, specifically within a function's header and exist for as long as their containing function is executing. A global variable (a variable with global scope) is defined outside all functions, exists for as long as its containing Python program is executing, and can be used within all functions in its containing Python program.

Python Variable Scope

	Local	**Global**
Where to Define	Inside a function	Outside all functions
Owner	The function where the variable is defined	The Python file where the variable is defined
Lifetime	Only as long as its containing function is executing	As long as its containing program is executing
Where Usable	Only inside the function where it is defined	In all functions of the Python program

The following Python code example contains variables with local and global scope. The variable *nShapes* is global because it is defined outside of all functions. Because it is a global variable, the code in the body of all functions may use the variable *nShapes*. Within the `square_area` function, the parameter named *length* and the variable named *area* both have local scope. Within the `rectangle_area` function, the parameters named *width* and *length* have local scope and the variable named *area* has local scope.

```
nShapes = 0

def square_area(length):
    area = length * length
    return area

def rectangle_area(width, length):
```

```python
    area = width * length
    return area
```

Because local variables are visible only within the function where they are defined, a programmer can define two variables with the same name as long as he defines them in different functions. In the previous example, both of the `square_area` and `rectangle_area` functions contain a parameter named *length* and a variable named *area*. All four of these variables are entirely separate and do not conflict with each other in any way because the scope of each variable is local to the function where it is defined.

A common mistake that many programmers make is to assume that a local variable can be used inside other functions. For example, the following Python code includes two functions named `main` and `circle_area`. Line 4 in `main` defines a variable named *radius*. Some programmers assume that the variable *radius* that is defined in `main` (and is therefore local to `main` only) can be used in `circle_area`. However, local variables from one function cannot be used inside another function. The local variables from `main` cannot be used inside `circle_area`.

```python
 1  import math
 2
 3  def main():
 4      radius = float(input("Enter the radius of a circle: "))
 5      area = circle_area()
 6      print(area)
 7
 8  def circle_area():
 9      # Mistake! There is no variable named radius
10      # defined inside this function, so the variable
11      # radius cannot be used in this function.
12      area = math.pi * radius * radius
13      return area
14
15  main()
```

Default Parameter Values and Optional Arguments

Python allows function parameters to have default values. If a parameter has a default value, then its corresponding argument is optional. If a function is called without an argument, the corresponding parameter gets its default value. Consider this example Python code:

```python
import math

def main():
    # Call the arc_length function with only one argument
    # even though the arc_length function has two parameters.
    len1 = arc_length(4.7)
    print(len1)

    # Call the arc_length function again but
    # this time with two arguments.
    len2 = arc_length(6.2, 270)
    print(len2)


# Define a function with two parameters. The
# second parameter has a default value of 360.
def arc_length(radius, degrees=360):
    """Compute and return the length of an arc of a circle."""
    circumference = 2 * math.pi * radius
    length = circumference * degrees / 360
    return length
```

```
22
23
24  main()
```

Notice at line 17 in the header for the `arc_length` function, that the parameter *radius* does not have a default value but the parameter *degrees* has a default value of 360. This means that when a programmer writes code to call the `arc_length` function, the programmer must pass a value for *radius* but is not required to pass a value for *degrees*. At line 6, the programmer wrote code to call the `arc_length` function and passed 4.7 for the *radius* parameter but did not pass a value for the *degrees*, so during that call to `arc_length`, the value of *degrees* will be the default value from line 17, which is 360. At line 11, the programmer wrote code to call the `arc_length` function again and passed two arguments: 6.2 and 270, so during that call to `arc_length`, the value of *degrees* will be 270.

Function Design

What are the properties of a good function?

There are many things to consider when writing a function, and many authors have written about design concepts that make functions easier to understand and less error prone. In future courses at BYU-Idaho, you will study some of these design concepts. For CSE 111, the following list contains a few properties that you should incorporate into your functions.

- A good function is understandable by other programmers. One way to make a function understandable is to write a documentation string at the top of the function that describes the function, its parameters, and its return value and to write comments in the body of the function as needed.
- A good function performs a single task that the programmer can describe, and the function's name matches its task.
- A good function is relatively short, perhaps fewer than 20 lines of code.
- A good function has as few decision points (`if` statements and loops) as possible. Too many decision points in a function make a function error prone and difficult to test.
- A good function is as reusable as possible. Functions that use parameters and return a result are more reusable than functions that get input from a user and print results to a terminal.

As you read the sample code in CSE 111, observe how the sample functions fit these good properties, and as you write programs for CSE 111, do your best to write functions that have these good properties.

[+]