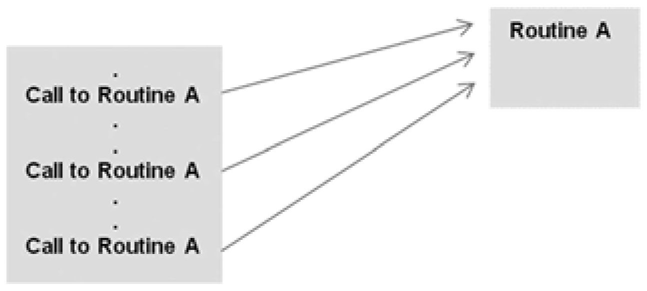
Program Routines and Functions

You have already been using Python's built-in functions such as `len`, `range`, and others. You will now look more closely at how functions are used in Python, as well as how to define our own.

What Is a Function Routine?

A **routine** is a named group of instructions performing some task. A routine can be **invoked** (*called*) as many times as needed in a given program, as shown below:

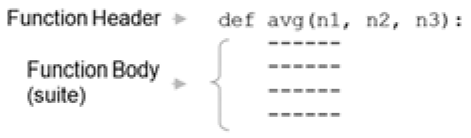


When a routine terminates, execution automatically returns to the point from which it was called. Such routines may be predefined in the programming language, or designed and implemented by the programmer.

A **function** is Python's version of a program routine. Some functions are designed to return a value, while others are designed for other purposes.

Defining Functions

In addition to the built-in functions of Python, there is the capability to define new functions. Such functions may be generally useful, or specific to a particular program. The elements of a function definition are given below:



```
def avg(n1, n2, n3):
    ------
    ------
    ------
    ------
```

The first line of a function definition is the *function header*. A function header starts with the keyword `def`, followed by an identifier (`avg`), which is the function's name. The function name is followed by a comma-separated (possibly empty) list of identifiers (`n1, n2, n3`) called **formal parameters**, or simply "parameters." Following the *parameter list* is a colon (**:**). Following the function header is the body of the function, a suite (program block) containing the function's instructions. As with all suites, the statements must be indented at the same level, relative to the function header.

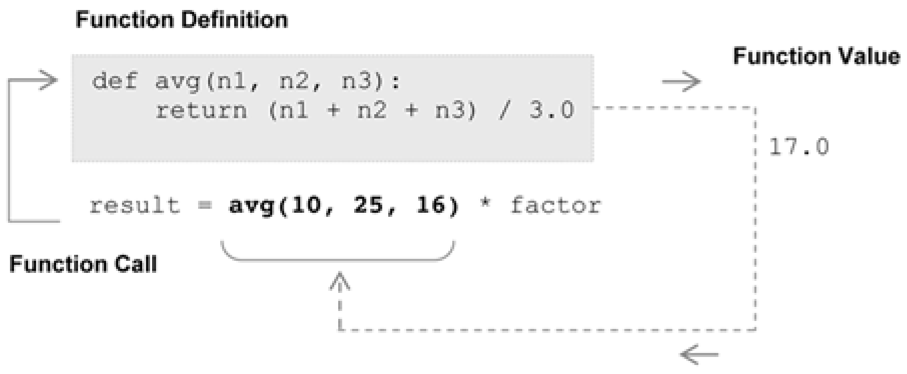
The number of items in a parameter list indicates the number of values that must be passed to the function, called **actual arguments** (or simply "arguments"), such as the variables `num1`, `num2`, and `num3` below.

```
>>> num1 = 10
>>>num2 = 25
>>>num3 = 16

>>>avg(num1,num2,num3)
```

Functions are generally defined at the top of a program. However, every function must be defined before it is called.

Value-Returning Functions

A **value-returning function** is a program routine called for its return value, and is therefore similar to a mathematical function. Take the simple mathematical function `f(x) 5 2x`. In this notation, "x" stands for any numeric value that function `f` may be applied to, for example, `f(2) 5 2x 5 4`. Program functions are similarly used, as illustrated below:



```
def avg(n1, n2, n3):
    return (n1 + n2 + n3) / 3.0

result = avg(10, 25, 16) * factor
```

Function `avg` takes three arguments (`n1`, `n2`, and `n3`) and returns the average of the three. The *function call* `avg(10, 25, 16)`, therefore, is an expression that evaluates to the returned function value. This is indicated in the function's *return statement* of the form `return` *`expr`*, where *`expr`* may be any expression.

Your Turn

From the Python Shell, first enter the following function, making sure to indent the code as given. Hit return twice after the last line of the function is entered. Then enter the following function calls and observe the results.

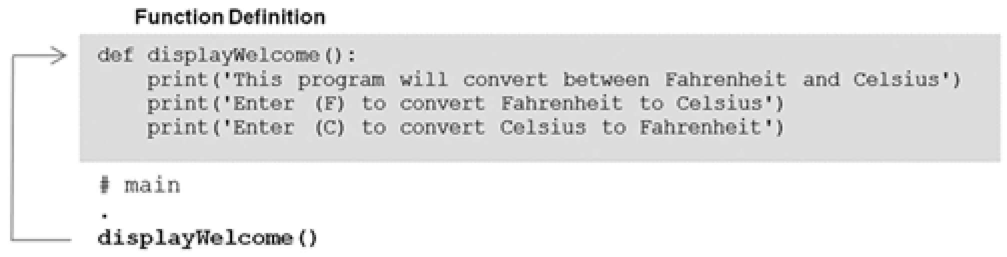
```
<<< def avg(n1, n2, n3):                  <<<avg(40, 10, 25)
        return (n1 + n2 + n3) / 3.0        ???

<<<avg(10, 25, 40)                        <<<avg(40, 25, 10)
???                                       ???
```

Part II - Non-Value-Returning Functions

A **non-value-returning function** is called not for a returned value, but for its *side effects*. A **side effect** is an action other than returning a function value, such as displaying output on the screen. There is a fundamental difference in the way that value-returning and non-value-returning functions are called. A call to a value-returning function is an expression, as for the call to function

`avg: result = ` **`avg(10, 25, 16)`** `* factor.`

When non-value-returning functions are called, however, the function call is a *statement*, as shown below. Since such functions do not have a return value, it is incorrect to use a call to a non-value-returning function as an expression.



```
def displayWelcome():
    print('This program will convert between Fahrenheit and Celsius')
    print('Enter (F) to convert Fahrenheit to Celsius')
    print('Enter (C) to convert Celsius to Fahrenheit')

# main
.
displayWelcome()
```

In this example, function `displayWelcome` is called only for the side-effect of the screen output produced. Finally, every function in Python is technically a value-returning function since *any function that does not explicitly return a function value (via a return statement) automatically returns the special value* None. We will, however, consider such functions as non-value-returning functions.

Your Turn

From the Python Shell, first enter the following function, making sure to indent the code as given. Then enter the following function calls and observe the results.

```
>>>def hello(name):                          >>> name = 'John'
       print('Hello', name + '!')                   >>>hello(name)
                                              ???
```

Self-Test Questions

1. The values passed in a given function call in Python are called,
  a) formal parameters
  b) actual arguments

2. The identifiers of a given Python function providing names for the values passed to it are called,
  a) formal parameters
  b) actual arguments

3. Functions can be called as many times as needed in a given program. (TRUE/FALSE)

4. When a given function is called, it is said to be,
  a) subrogated
  b) invoked
  c) activated

5. Which of the following types of functions must contain a return statement,
  a) value-returning functions
  b) non-value-returning functions

6. Value-returning function calls are,

a) expressions
b) statements

7. Non-value-returning function calls are,
a) expressions
b) statements

8. Which of the following types of routines is meant to produce side effects?
a) value-returning functions
b) non-value-returning functions

Problem Solving

1. Function `avg` returns the average of three values. Which of the following statements, each making calls to function `avg`, are valid? (Assume that all variables are of numeric type.)
a) `result = avg(n1, n2)`
b) `result = avg(n1, n2, avg(n3, n4, n5))`
c) `result = avg(n1 + n2, n3 + n4, n5 + n6)`
d) `print(avg(n1, n2, n3))`
e) `avg(n1, n2, n3)`

2. Which of the following statements, each involving calls to function `displayWelcome` displaying a welcome message on the screen, are valid?
a) `print(displayWelcome)`
b) `displayWelcome`
c) `result = displayWelcome()`
d) `displayWelcome()`

3. Write a Python function named `zeroCheck` that is given three integers, and returns true if any of the integers is 0, otherwise it returns false.

4. Write a Python function named `ordered3` that is passed three integers, and returns true if the three integers are in order from smallest to largest, otherwise it returns false.

5. Write a Python function named `modCount` that is given a positive integer, n, and a second positive integer, `m <= n`, and returns how many numbers between 1 and n are evenly divisible by m.