Value Returning Functions and Modules

Addison-Wesley
is an imprint of
PEARSON

Topics

- Introduction to Value-returning Functions: Generating Random Numbers
- Writing Your Own Value-Returning Functions
- The `math` Module
- Storing Functions in Modules

Introduction to Value-Returning Functions: Generating Random Numbers

- <u>Simple function</u>: **group of statements within a program for performing a specific task**
  - Call function when you need to perform the task
- <u>Value-returning function</u>: **similar to simple function, returns a value**
  - Value returned to part of program that called the function when function finishes executing

Standard Library Functions and the `import` Statement

- Standard library: library of pre-written functions that comes with Python
  - *Library functions* perform tasks that programmers commonly need
    - Example: `print`, `input`, `range`
    - Viewed by programmers as a “black box”
- Some library functions built into Python interpreter
  - To use, just call the function

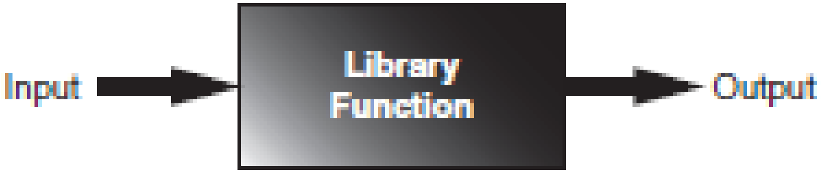
Addison-Wesley
is an imprint of
PEARSON

Standard Library Functions and the `import` Statement (cont'd.)

- **Modules: files that stores functions of the standard library**
  - Help organize library functions not built into the interpreter
  - Copied to computer when you install Python
- **To call a function stored in a module, need to write an `import` statement**
  - Written at the top of the program
  - Format: ***`import module_name`***

Standard Library Functions and the `import` Statement (cont’d.)

Figure 6-1 A library function viewed as a black box



Generating Random Numbers

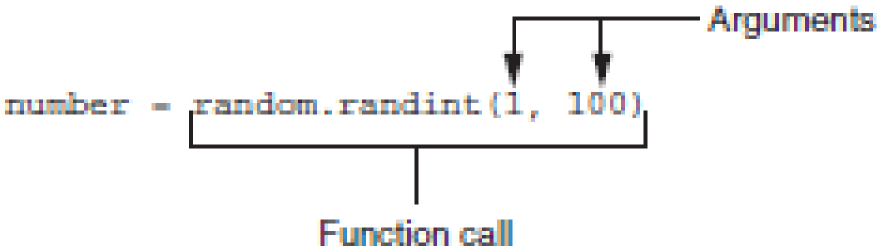
- Random number are useful in a lot of programming tasks
- `random` module: includes library functions for working with random numbers
- Dot notation: notation for calling a function belonging to a module
  - Format: `module_name.function_name()`

Generating Random Numbers (cont'd.)

- **`randint` function: generates a random number in the range provided by the arguments**
  - Returns the random number to part of program that called the function
  - Returned integer can be used anywhere that an integer would be used
  - You can experiment with the function in interactive mode

Generating Random Numbers (cont'd.)

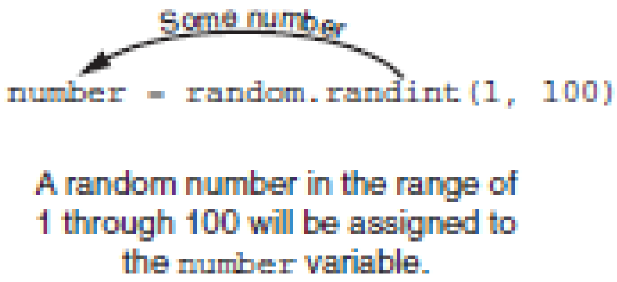
Figure 6-2 A statement that calls the `random` function



```
number = random.randint(1, 100)
```

Generating Random Numbers (cont'd.)

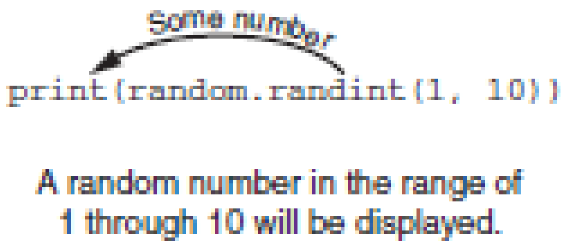
Figure 6-3 The `random` function returns a value



```
number = random.randint(1, 100)
```

A random number in the range of
1 through 100 will be assigned to
the `number` variable.

Figure 6-4 Displaying a random number



```
print(random.randint(1, 10))
```

A random number in the range of
1 through 10 will be displayed.

Addison-Wesley
is an imprint of

PEARSON

Random Number Seeds

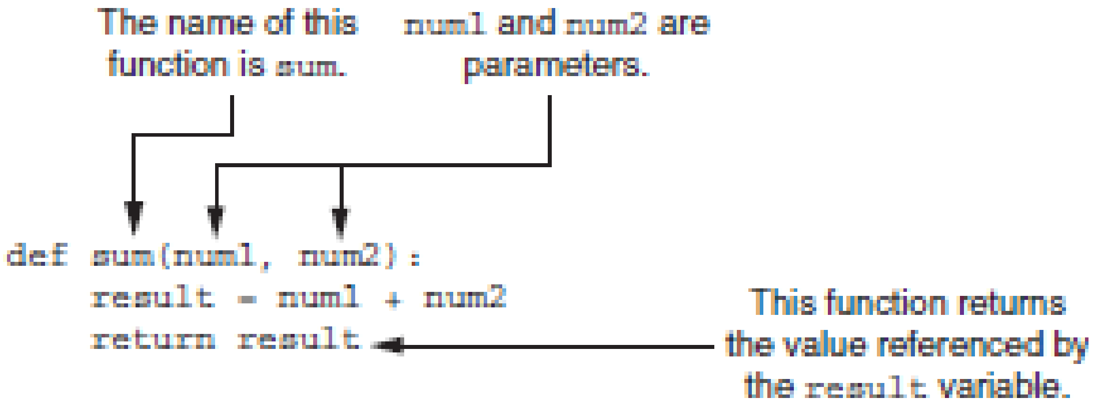
- **Random number created by functions in random module are actually pseudo-random numbers**
- **<u>Seed value</u>: initializes the formula that generates random numbers**
  - Need to use different seeds in order to get different series of random numbers
    - By default uses system time for seed
    - Can use `random.seed()` function to specify desired seed value

Writing Your Own Value-Returning Functions

- **To write a value-returning function, you write a simple function and add one or more `return` statements**
  - Format: `return` *`expression`*
    - The value for *`expression`* will be returned to the part of the program that called the function
  - The expression in the `return` statement can be a complex expression, such as a sum of two variables or the result of another value-returning function

Writing Your Own Value-Returning Functions (cont’d.)

Figure 6-5 Parts of the function



The name of this function is `sum`.

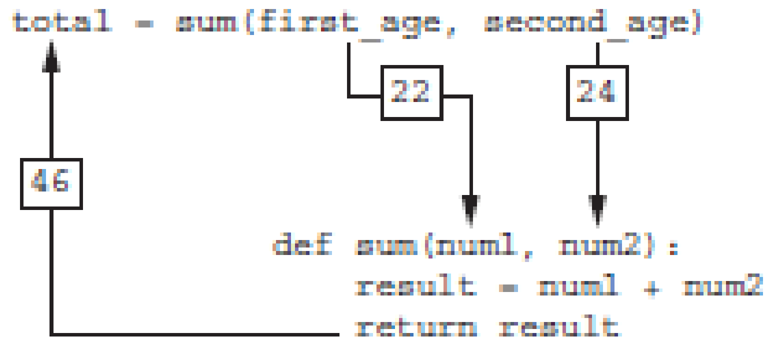
`num1` and `num2` are parameters.

```
def sum(num1, num2):
    result = num1 + num2
    return result
```

This function returns the value referenced by the `result` variable.

Writing Your Own Value-Returning Functions (cont’d.)

Figure 6-6 Arguments are passed to the sum function and a value is returned



```
total = sum(first_age, second_age)

def sum(num1, num2):
    result = num1 + num2
    return result
```

How to Use Value-Returning Functions

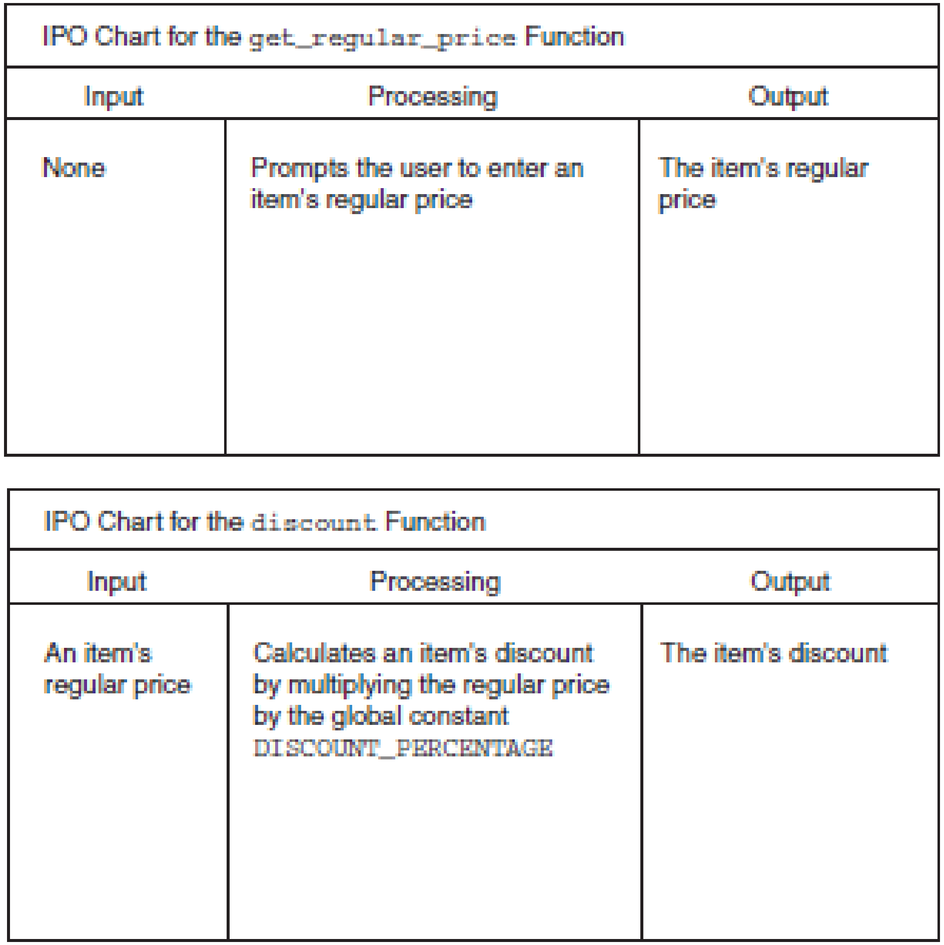
- **Value-returning function can be useful in specific situations**
  - Example: have function prompt user for input and return the user’s input
  - Simplify mathematical expressions
  - Complex calculations that need to be repeated throughout the program
- **Use the returned value**
  - Assign it to a variable or use as an argument in another function

Using IPO Charts

- **IPO chart: describes the input, processing, and output of a function**
  - Tool for designing and documenting functions
  - Typically laid out in columns
  - Usually provide brief descriptions of input, processing, and output, without going into details
    - Often includes enough information to be used instead of a flowchart

Using IPO Charts (cont’d.)

Figure 6-7 IPO charts for the getRegularPrice and discount functions



IPO Chart for the get_regular_price Function

Input	Processing	Output
None	Prompts the user to enter an item's regular price	The item's regular price

IPO Chart for the discount Function

Input	Processing	Output
An item's regular price	Calculates an item's discount by multiplying the regular price by the global constant DISCOUNT_PERCENTAGE	The item's discount

Returning Boolean Values

- **Boolean function: returns either `True` or `False`**
  - Use to test a condition such as for decision and repetition structures
    - Common calculations, such as whether a number is even, can be easily repeated by calling a function
  - Use to simplify complex input validation code

Returning Multiple Values

- **In Python, a function can return multiple values**
  - Specified after the return statement separated by commas
    - Format: `return` *`expression1,`*
      `expression2, etc.`
  - When you call such a function in an assignment statement, you need a separate variable on the left side of the = operator to receive each returned value

Addison-Wesley
is an imprint of
PEARSON

The `math` Module

- **`math` module: part of standard library that contains functions that are useful for performing mathematical calculations**
  - Typically accept one or more values as arguments, perform mathematical operation, and return the result
  - Use of module requires an `import math` statement

The `math` Module (cont’d.)

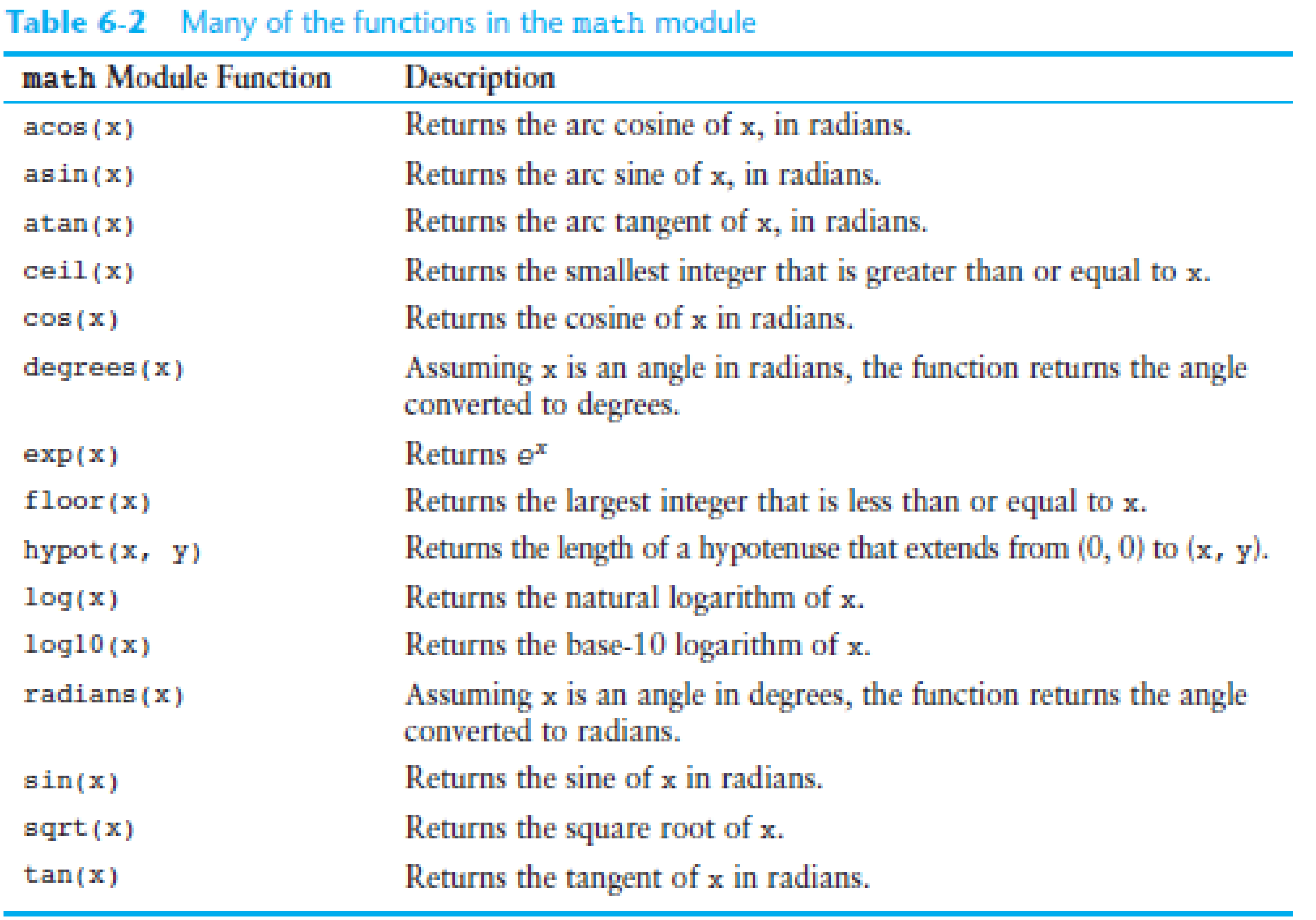


Table 6-2 Many of the functions in the `math` module

`math` Module Function	Description
`acos(x)`	Returns the arc cosine of `x`, in radians.
`asin(x)`	Returns the arc sine of `x`, in radians.
`atan(x)`	Returns the arc tangent of `x`, in radians.
`ceil(x)`	Returns the smallest integer that is greater than or equal to `x`.
`cos(x)`	Returns the cosine of `x` in radians.
`degrees(x)`	Assuming `x` is an angle in radians, the function returns the angle converted to degrees.
`exp(x)`	Returns e^x
`floor(x)`	Returns the largest integer that is less than or equal to `x`.
`hypot(x, y)`	Returns the length of a hypotenuse that extends from (0, 0) to (`x`, `y`).
`log(x)`	Returns the natural logarithm of `x`.
`log10(x)`	Returns the base-10 logarithm of `x`.
`radians(x)`	Assuming `x` is an angle in degrees, the function returns the angle converted to radians.
`sin(x)`	Returns the sine of `x` in radians.
`sqrt(x)`	Returns the square root of `x`.
`tan(x)`	Returns the tangent of `x` in radians.

The `math` Module (cont'd.)

- **The `math` module defines variables `pi` and `e`, which are assigned the mathematical values for *pi* and *e***
  - Can be used in equations that require these values, to get more accurate results
- **Variables must also be called using the dot notation**
  - Example:

```
circle_area = math.pi * radius**2
```

Addison-Wesley
is an imprint of

PEARSON

Storing Functions in Modules

- **In large, complex programs, it is important to keep code organized**
- **Modularization: grouping related functions in modules**
  - Makes program easier to understand, test, and maintain
  - Make it easier to reuse code for multiple different programs
    - Import the module containing the required function to each program that needs it

Addison-Wesley is an imprint of PEARSON

Storing Functions in Modules (cont'd.)

- **Module is a file that contains Python code**
  - Contains function definition but does not contain calls to the functions
    - **Importing** programs will call the functions
- **Rules for module names:**
  - File name should end in `.py`
  - Cannot be the same as a Python keyword
- **Import module using `import` statement**

Addison-Wesley is an imprint of PEARSON

Menu Driven Programs

- **<u>Menu-driven program</u>: displays a list of operations on the screen, allowing user to select the desired operation**
  - List of operations displayed on the screen is called a *menu*
- **Program uses a decision structure to determine the selected menu option and required operation**
- **Typically repeats in loop till user quits**

Summary

- **This chapter covered:**
  - Value-returning functions, including:
    - Writing value-returning functions
    - Using value-returning functions
    - Functions returning multiple values
  - Using library functions and the `import` statement
  - Modules, including:
    - The `random` and `math` modules
    - Grouping your own functions in modules

Practice Exercises using Python command line

```
Import random
number = random.randint(1,100)
print(number)
```

(use the up arrow key to re-issue the last two commands and see a new random generated number; replace the upper limit 100 with a different number, 10 for instance)

```
Import random
number = random.randint(1,10)
print(number)
```

Practice Exercise – create and call a function

```
def sum(num1, num2):
    result = num1 + num2
    return result

total = sum(10,20)
print(total)

n1 = 10
n2 = 20
total = sum(n1, n2)
print(total)

n1 = float(input('Enter the first number: '))
n2 = float(input('Enter the second number: '))
total = sum(n1, n2)
print(total)
```

Practice Exercise – use a math function

```
import math
radius =2

circle_area = math.pi * radius**2

print(circle_area)
```

Practice Exercise – use a random function

```
#this program displays 5 random numbers in the range 1 thru 100
import  random
def main():
    for count in range(5):
        #Get a random number
        number = random.randint(1,100)
        #Display the number
        print(number)

#Call the main function
main(
```

Note: you can experiment with random numbers in interactive mode

```
>>> import random
>>> random.randint(1,100)
5
>>> random.randint(1,200)
98
>>> random.randint(100,200)
181
>>>
```

Practice Exercise – use a random function (cont)

```
>>> import random
>>> random.seed(10)
>>> random.randint(1,100)
74
>>> random.randint(1,100)
5
>>>
```

Note: notice the same sequence after you issue random.seed(10) again

```
>>> random.seed(10)
>>> random.randint(1,100)
74
>>> random.randint(1,100)
5
>>>
```

Practice Exercise – Storing Functions in Modules

Create, test and save this program as circle.py

```
#this script will saved (will be behave as a module)
#be imported later into another script

import math
def area(radius):
    return math.pi * radius**2

def circumference(radius):
    return 2 * math.pi * radius
```

Create, test and save this program geometry.py

```
#This program imports the circle.py module created earlier
import circle
c = circle.area(2)
print(c)
```