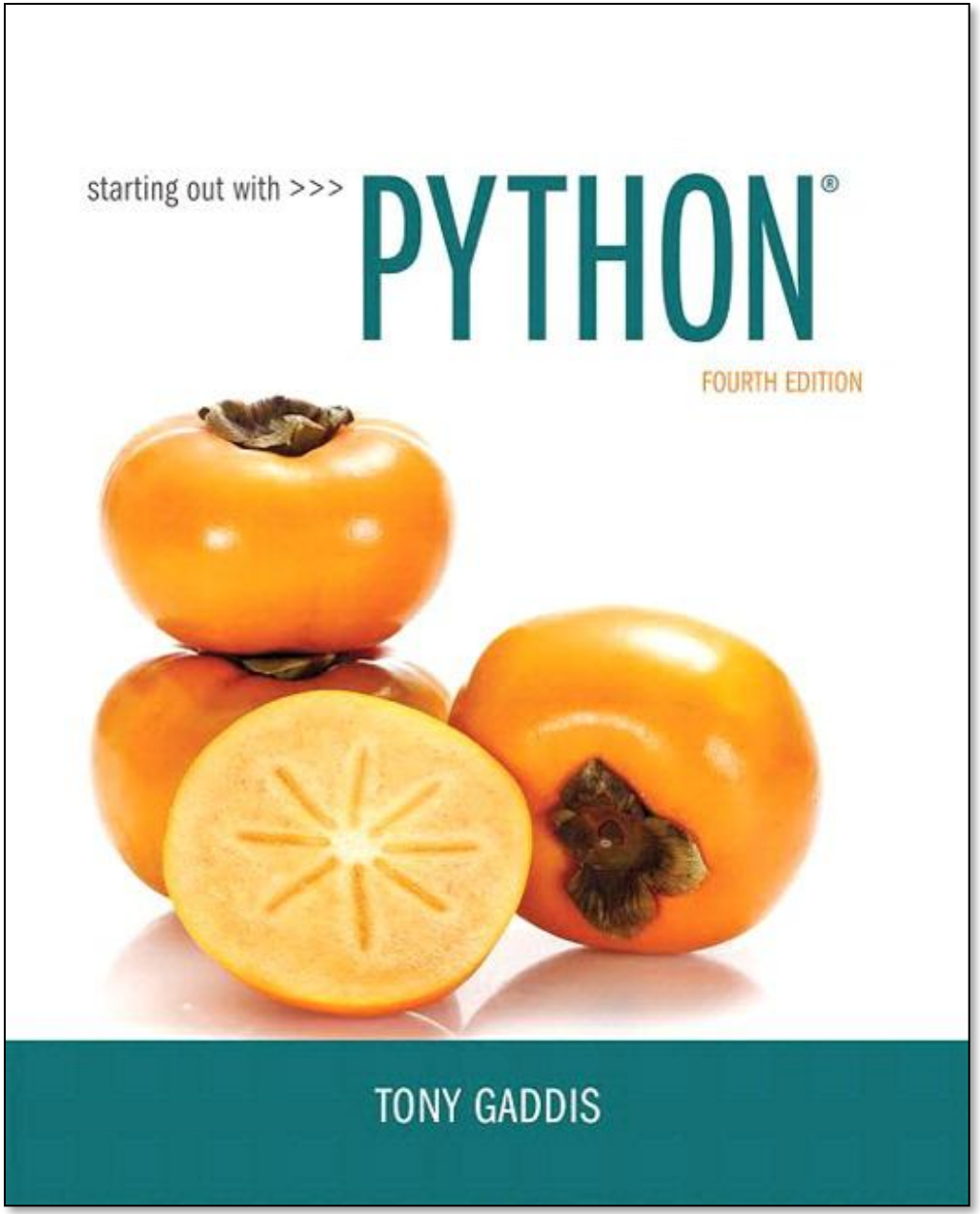
CHAPTER 5

Functions



Topics

- Introduction to Functions
- Defining and Calling a Void Function
- Designing a Program to Use Functions
- Local Variables
- Passing Arguments to Functions
- Global Variables and Global Constants
- Turtle Graphics: Modularizing Code with Functions

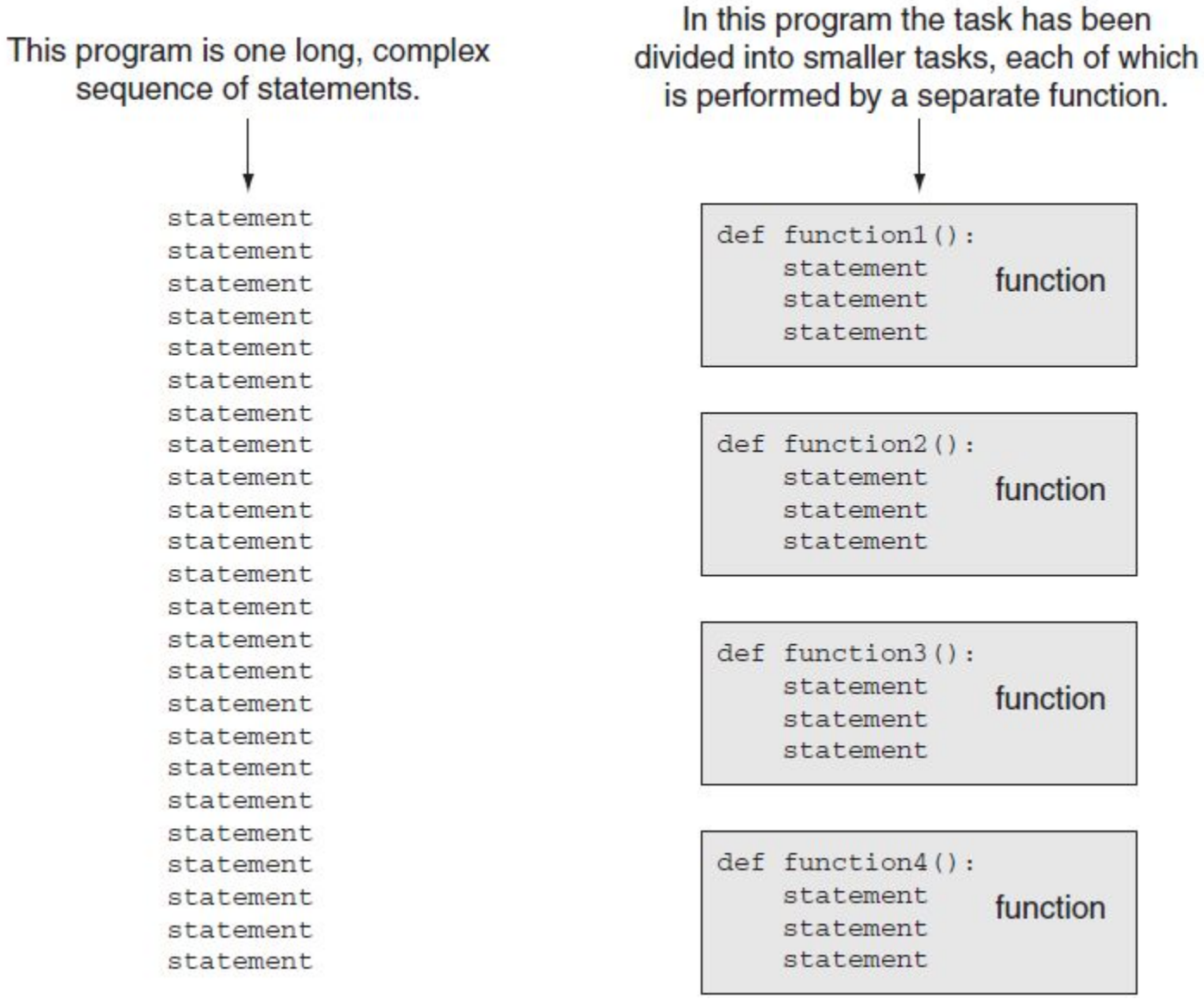
Topics (cont'd.)

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

Introduction to Functions

- **Function: group of statements within a program that perform as specific task**
  - Usually one task of a large program
    - Functions can be executed in order to perform overall program task
  - Known as *divide and conquer* approach
- **Modularized program: program wherein each task within the program is in its own function**

Figure 5-1 Using functions to divide and conquer a large task



Benefits of Modularizing a Program with Functions

- **The benefits of using functions include:**
  - Simpler code
  - Code reuse
    - write the code once and call it multiple times
  - Better testing and debugging
    - Can test and debug each function individually
  - Faster development
  - Easier facilitation of teamwork
    - Different team members can write different functions

Void Functions and Value-Returning Functions

- **A <u>void function</u>:**
  - Simply executes the statements it contains and then terminates.
- **A <u>value-returning function</u>:**
  - Executes the statements it contains, and then it returns a value back to the statement that called it.
    - The `input`, `int`, and `float` functions are examples of value-returning functions.

Defining and Calling a Function

- **Functions are given names**
  - Function naming rules:
    - Cannot use key words as a function name
    - Cannot contain spaces
    - First character must be a letter or underscore
    - All other characters must be a letter, number or underscore
    - Uppercase and lowercase characters are distinct

Defining and Calling a Function (cont'd.)

- **Function name should be descriptive of the task carried out by the function**
  - Often includes a verb
- **Function definition: specifies what function does**

```
def function_name():
        statement
        statement
```

Defining and Calling a Function (cont'd.)

- **Function header: first line of function**
  - Includes keyword `def` and function name, followed by parentheses and colon
- **Block: set of statements that belong together as a group**
  - Example: the statements included in a function

Defining and Calling a Function (cont'd.)

- **Call a function to execute it**
  - When a function is called:
    - Interpreter jumps to the function and executes statements in the block
    - Interpreter jumps back to part of program that called the function
      - Known as function return

Defining and Calling a Function (cont'd.)

- **`main` function: called when the program starts**
  - Calls other functions when they are needed
  - Defines the *mainline logic* of the program

Indentation in Python

- **Each block must be indented**
  - Lines in block must begin with the same number of spaces
    - Use tabs or spaces to indent lines in a block, but not both as this can confuse the Python interpreter
    - IDLE automatically indents the lines in a block
  - Blank lines that appear in a block are ignored

Designing a Program to Use Functions

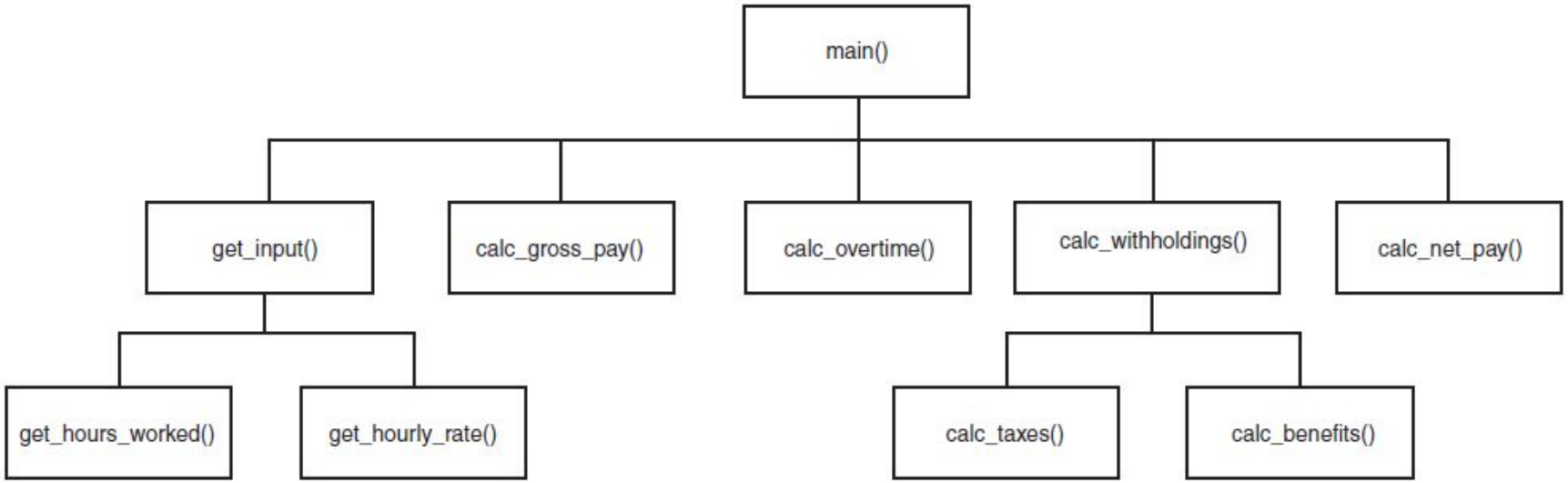
- **In a flowchart, function call shown as rectangle with vertical bars at each side**
  - Function name written in the symbol
  - Typically draw separate flow chart for each function in the program
    - End terminal symbol usually reads `Return`
- **Top-down design: technique for breaking algorithm into functions**

Designing a Program to Use Functions (cont'd.)

- **Hierarchy chart: depicts relationship between functions**
  - AKA structure chart
  - Box for each function in the program, Lines connecting boxes illustrate the functions called by each function
  - Does not show steps taken inside a function
- **Use `input` function to have program wait for user to press enter**

Designing a Program to Use Functions (cont'd.)

Figure 5-10 A hierarchy chart



Time to Practice

- **Turn to page 219**
- **Read the "In the Spotlight" – Defining and Calling Functions**
  - Class discussions for code
- **Students create the acme.dryer.py program**
- **Print the Output only and submit it to the basket**

Local Variables

- **Local variable: variable that is assigned a value inside a function**
  - Belongs to the function in which it was created
    - Only statements inside that function can access it, error will occur if another function tries to access the variable
- **Scope: the part of a program in which a variable may be accessed**
  - For local variable: function in which created

Local Variables (cont'd.)

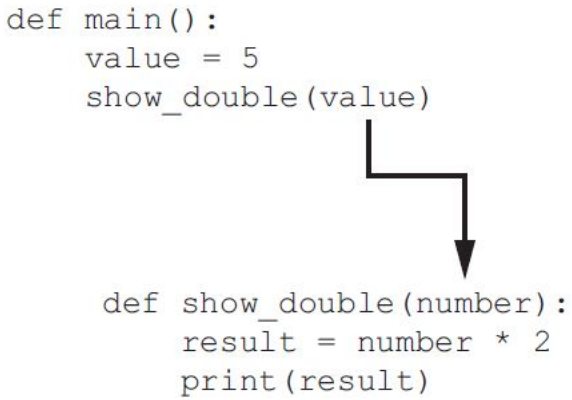
- **Local variable cannot be accessed by statements inside its function which precede its creation**
- **Different functions may have local variables with the same name**
  - Each function does not see the other function's local variables, so no confusion

Passing Arguments to Functions

- **Argument: piece of data that is sent into a function**
  - Function can use argument in calculations
  - When calling the function, the argument is placed in parentheses following the function name

Passing Arguments to Functions (cont'd.)

Figure 5-13 The `value` variable is passed as an argument



Passing Arguments to Functions (cont'd.)

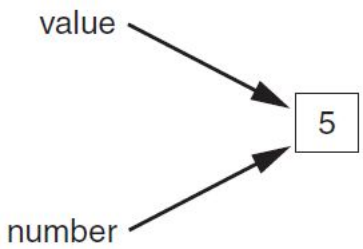
- **Parameter variable: variable that is assigned the value of an argument when the function is called**
  - The parameter and the argument reference the same value
  - General format:
  - `def function_name(parameter):`
  - Scope of a parameter: the function in which the parameter is used

Passing Arguments to Functions (cont'd.)

Figure 5-14 The `value` variable and the `number` parameter reference the same value

```
def main():
    value = 5
    show_double(value)


def show_double(number):
    result = number * 2
    print(result)
```



Time to Practice

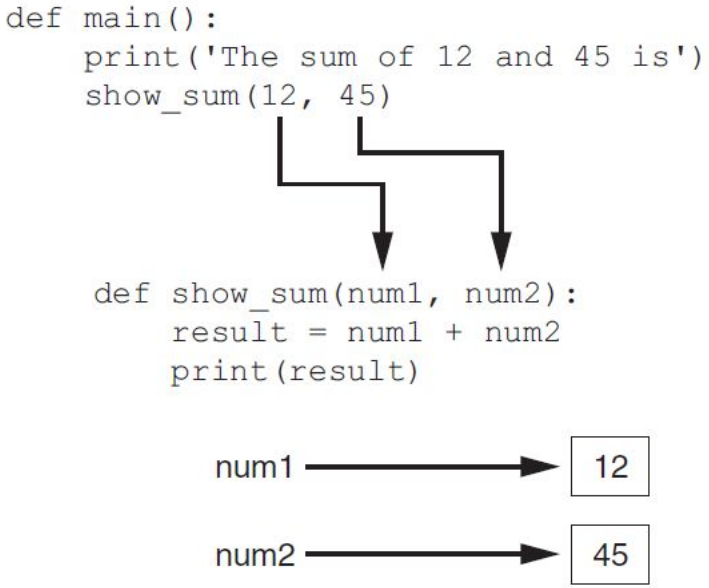
- **Turn to page 228**
- **Read the "In the Spotlight" – Passing an Argument to a Function**
  - Class discussions for code
- **Students create the cups_to_ounces.py program**
- **Print the Output only and submit it to the basket**

Passing Multiple Arguments

- **Python allows writing a function that accepts multiple arguments**
  - Parameter list replaces single parameter
    - Parameter list items separated by comma
- **Arguments are passed *by position* to corresponding parameters**
  - First parameter receives value of first argument, second parameter receives value of second argument, etc.

Passing Multiple Arguments (cont'd.)

Figure 5-16 Two arguments passed to two parameters



Making Changes to Parameters

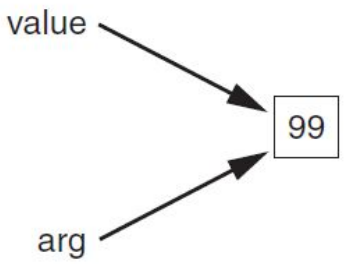
- **Changes made to a parameter value within the function do not affect the argument**
  - Known as *pass by value*
  - Provides a way for unidirectional communication between one function and another function
    - Calling function can communicate with called function

Making Changes to Parameters (cont'd.)

Figure 5-17 The value variable is passed to the `change_me` function

```
def main():
    value = 99
    print('The value is', value)
    change_me(value)
    print('Back in main the value is', value)


def change_me(arg):
    print('I am changing the value.')
    arg = 0
    print('Now the value is', arg)
```



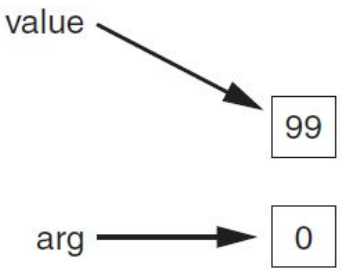
Making Changes to Parameters (cont'd.)

- **Figure 5-18**
  - The `value` variable passed to the `change_me` function cannot be changed by it

Figure 5-18 The `value` variable is passed to the `change_me` function

```
def main():
    value = 99
    print('The value is', value)
    change_me(value)
    print('Back in main the value is', value)


def change_me(arg):
    print('I am changing the value.')
    arg = 0
    print('Now the value is', arg)
```



Keyword Arguments

- **Keyword argument: argument that specifies which parameter the value should be passed to**
  - Position when calling function is irrelevant
  - General Format:
  - `function_name(parameter=value)`
- **Possible to mix keyword and positional arguments when calling a function**
  - Positional arguments must appear first

Global Variables and Global Constants

- **Global variable: created by assignment statement written outside all the functions**
  - Can be accessed by any statement in the program file, including from within a function
  - If a function needs to assign a value to the global variable, the global variable must be redeclared within the function
    - General format: `global variable_name`

Global Variables and Global Constants (cont'd.)

- **Reasons to avoid using global variables:**
  - Global variables making debugging difficult
    - Many locations in the code could be causing a wrong variable value
  - Functions that use global variables are usually dependent on those variables
    - Makes function hard to transfer to another program
  - Global variables make a program hard to understand

Global Constants

- **Global constant: global name that references a value that cannot be changed**
  - Permissible to use global constants in a program
  - To simulate global constant in Python, create global variable and do not re-declare it within functions

Time to Practice

- **Turn to page 237**
- **Read the "In the Spotlight" – Using Global Constants**
  - Class discussions for code
- **Students create the retirement.py program**
- **Print the Output only and submit it to the basket**

Introduction to Value-Returning Functions: Generating Random Numbers

- **void function: group of statements within a program for performing a specific task** (talked about previously)
  - Call function when you need to perform the task
- **Value-returning function: similar to void function, returns a value** (talked about previously)
  - 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

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 5-19 A library function viewed as a black box

Generating Random Numbers

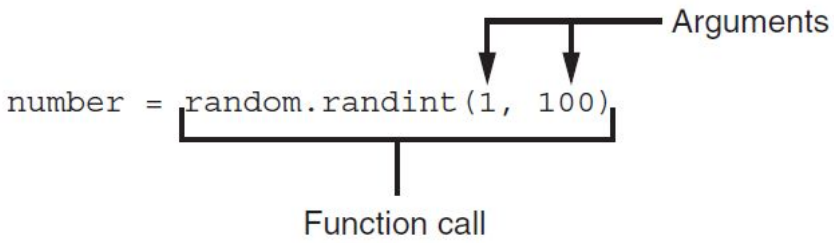
- **Random number are useful in a lot of programming tasks**
- **`random` module: includes library functions for working with random numbers**
- **Dot notation: notation for calling a function belonging to a module**
  - Format: `module_name.function_name()`

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
  - Arguments include begin and ending value

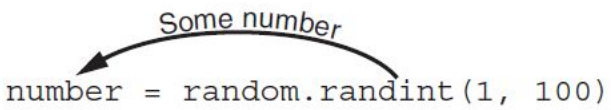
Generating Random Numbers (cont'd.)

Figure 5-20 A statement that calls the `random` function



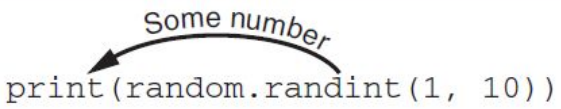
Generating Random Numbers (cont'd.)

Figure 5-21 The `random` function returns a value



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

Figure 5-22 Displaying a random number



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

Generating Random Numbers (cont'd.)

- **`randrange` function: similar to `range` function, but returns randomly selected integer from the resulting sequence**
  - Same arguments as for the `range` function
  - Argument starts at 0 and ends one less then end value.
- **`random` function: you do not pass any arguments to the random function, when called , it returns a random float in the range of 0.0 and 1.0.**
- **`uniform` function: returns a random float but allows user to specify range ex. 1.0, 10.0**

Random Number Seeds

- **Random number created by functions in random module are actually pseudo-random numbers**
- **Seed value: 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
    - Used in calculation that returns the next random number in a series. When the random module is imported, it retrieves the system time from the computer's internal clock and uses that as the seed value

Time to Practice

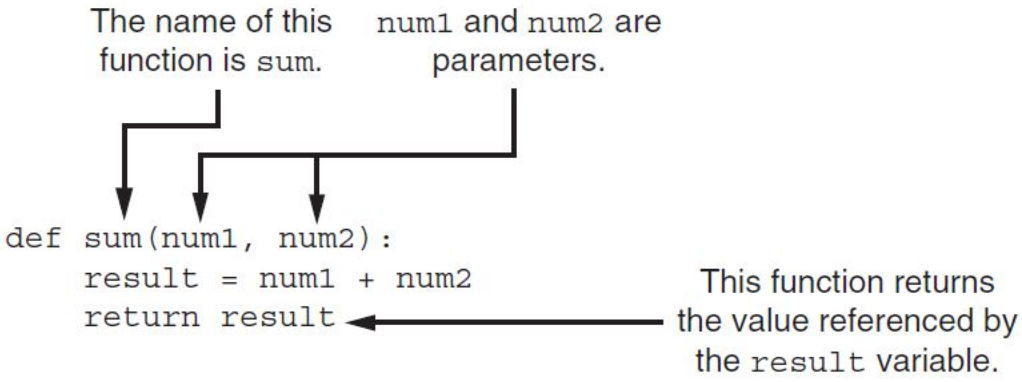
- **Turn to page 244**
- **Read the "In the Spotlight" – Using Random Numbers**
  - Class discussions for code
- **Students create the dice.py program**
- **Print the Output only and submit it to the basket**

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 5-23 Parts of the function



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 5-25 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 Strings

- **You can write functions that return strings**
- **For example:**

```
def get_name():
    # Get the user's name.
    name = input('Enter your name: ')
    # Return the name.
    return name
```

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

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 5-2 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
```

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

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**

Menu Driven Programs

- **Menu-driven program: 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 until the user quits

Time to Practice

- **Turn to page 255 - 258**
- **Read the "In the Spotlight" – Modularizing with Functions**
  - Class discussions for code
    - Program is divided into 4 sections
- **Students create the commission_rate.py program**
- **Make sure you call the main function**
  - Main ( )
- **Print the Output only and submit it to the basket**

Module Example

- **Your Instructor asked you to write a program that calculates the following:**
  - The area of a circle
  - The circumference of a circle
  - The area of a rectangle
  - The perimeter of a rectangle
- **Two categories of calculations required in this program: those related to circles, and those related to rectangles. You could write all of the circle-related functions in one module, and the rectangle-related functions in another module.**
- **Turn to page 264**

Module Example - cont

- **Program 5-26 – show the circle module, which contains 2 functions: area and circumference**
- **Program 5-27 – show the rectangle module, which contains 2 functions area and perimeter.**
- **To use the modules, you will need to import them, once it's imported you can call it's functions**
  - Import circle

Module Example – cont.

- **Program 5-28**
  - Line 4 and 5 – importing
  - Line 30, 34, and 43 (referencing the stored functions)
- **This program is a menu driven program, it displays a list of operations on the screeen**
- **GO to your computers and complete pages 264 - 268**

Summary

- **This chapter covered:**
  - The advantages of using functions
  - The syntax for defining and calling a function
  - Methods for designing a program to use functions
  - Use of local variables and their scope
  - Syntax and limitations of passing arguments to functions
  - Global variables, global constants, and their advantages and disadvantages

Summary (cont'd.)

- 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
- Modularizing Turtle Graphics Code