Chapter 3: Simple Functions
Topics

- Introduction to Functions
- Defining and Calling a Function
- Designing a Program to Use Functions
- Local Variables
- Passing Arguments to Functions
- Global Variables and Global Constants

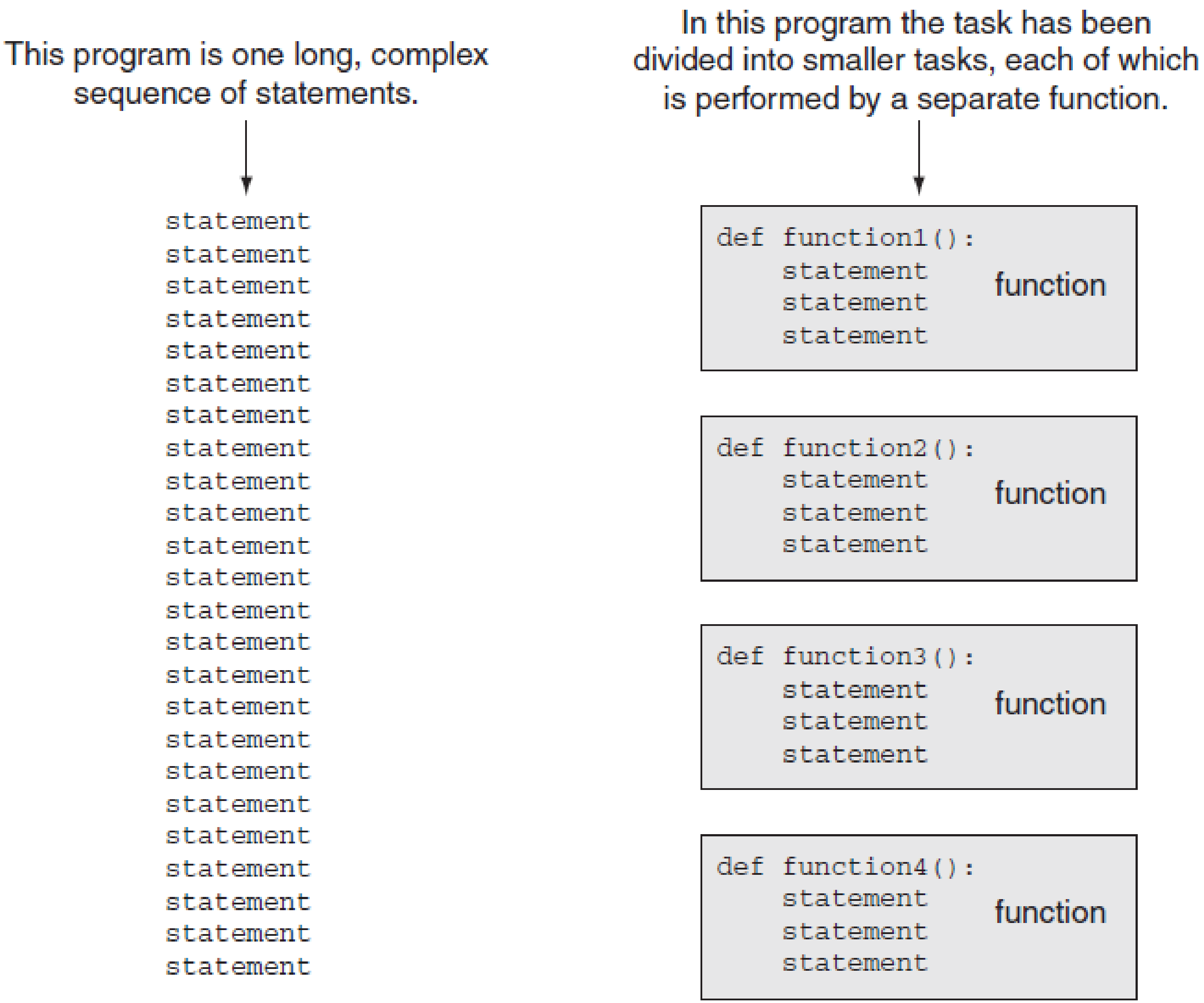
Addison-Wesley
is an imprint of
PEARSON

Introduction to Functions

- **<u>Function</u>: 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
- **<u>Modularized program</u>: program wherein each task within the program is in its own function**

Addison-Wesley
is an imprint of
PEARSON

Figure 3-1 Using functions to divide and conquer a large task



This program is one long, complex sequence of statements.

```
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
```

In this program the task has been divided into smaller tasks, each of which is performed by a separate function.

```
def function1():
    statement
    statement
    statement
```

function

```
def function2():
    statement
    statement
    statement
```

function

```
def function3():
    statement
    statement
    statement
```

function

```
def function4():
    statement
    statement
    statement
```

function

Addison-Wesley
is an imprint of
PEARSON

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

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

Addison-Wesley is an imprint of PEARSON

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

Addison-Wesley
is an imprint of
PEARSON

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`
- **<u>Top-down design</u>: technique for breaking algorithm into functions**

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

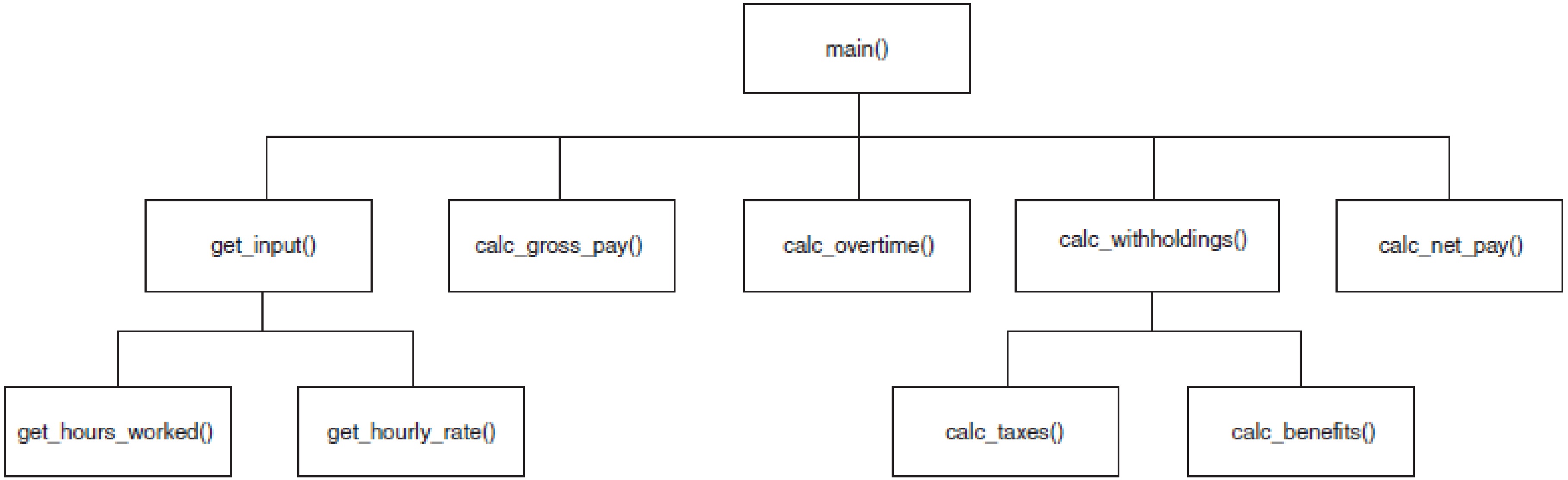
- **Hierarchy chart: depicts relationship between functions**
  - AKA structure chart
  - Box for each function in the program, Lines connecting boxes illustrate the functions called by each function
  - Does not show steps taken inside a function
- **Use `input` function to have program wait for user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

Addison-Wesley
is an imprint of

PEARSON

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

Figure 3-10 A hierarchy chart



Addison-Wesley
is an imprint of

PEARSON

Local Variables

- **<u>Local variable</u>: 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
- **<u>Scope</u>: 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

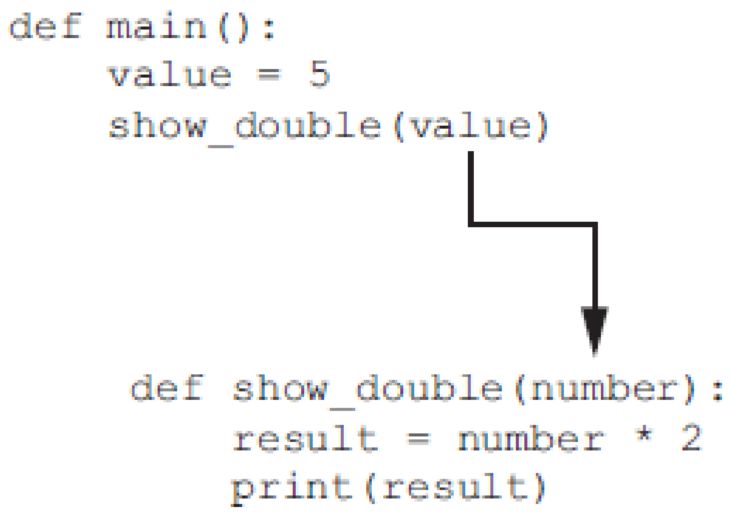
- **<u>Argument</u>: piece of data that is sent into a function**
  - Function can use argument in calculations
  - When calling the function, the argument is placed in parentheses following the function name

Addison-Wesley
is an imprint of

PEARSON

Passing Arguments to Functions (cont'd.)

Figure 3-13 The `value` variable is passed as an argument



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

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

Passing Arguments to Functions (cont'd.)

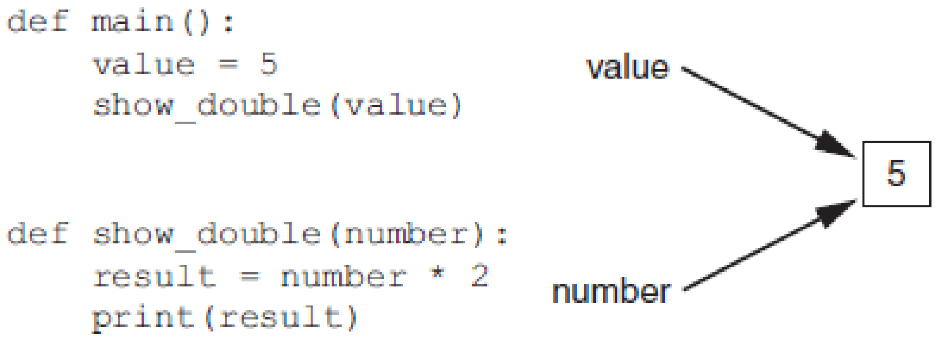
- **Parameter variable: variable that is assigned the value of an argument when the function is called**
  - The parameter and the argument reference the same value
  - General format:

    ```
    def function_name(parameter) :
    ```
  - Scope of a parameter: the function in which the parameter is used

Addison-Wesley
is an imprint of
PEARSON

Passing Arguments to Functions (cont'd.)

Figure 3-14 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)
```

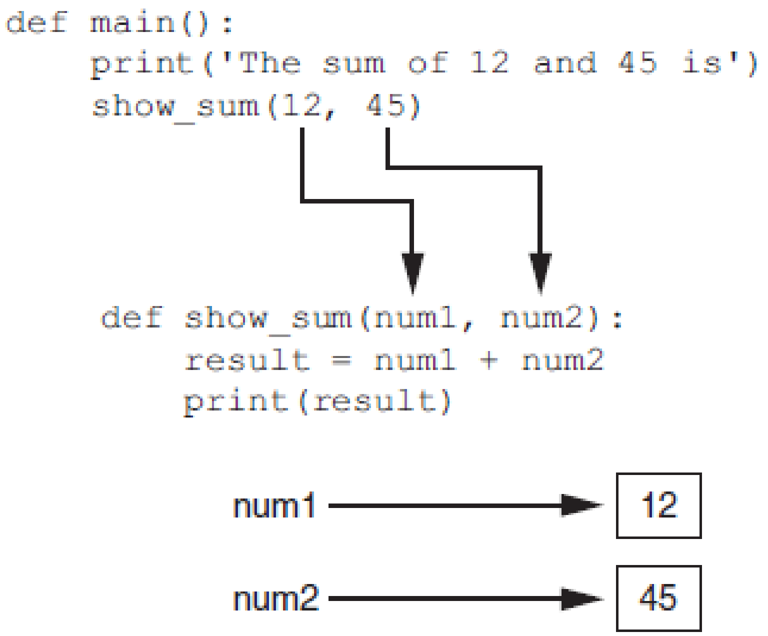
Addison-Wesley
is an imprint of
PEARSON

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 3-16 Two arguments passed to two parameters



```
def main():
    print('The sum of 12 and 45 is')
    show_sum(12, 45)

def show_sum(num1, num2):
    result = num1 + num2
    print(result)
```

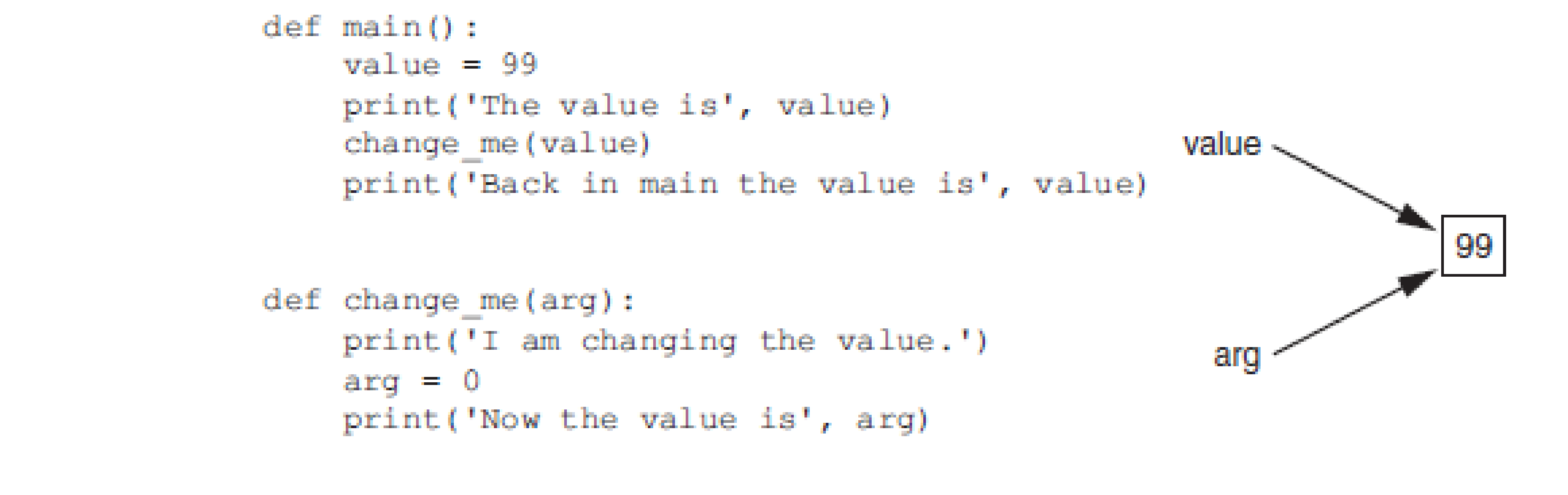
Making Changes to Parameters

- **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

Addison-Wesley
is an imprint of

PEARSON

Making Changes to Parameters (cont'd.)



```
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)
```

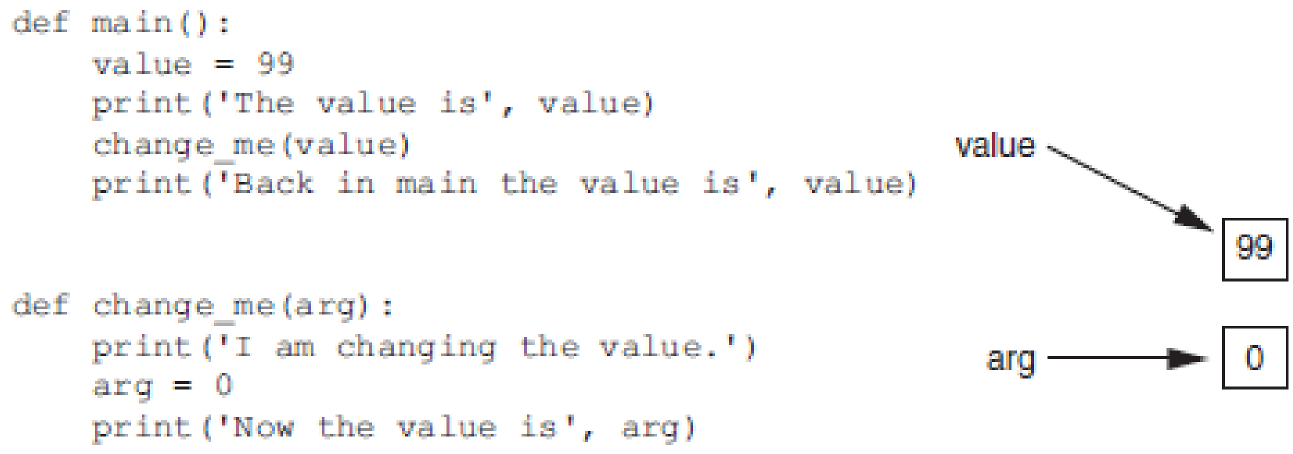
Figure 3-17 The value variable is passed to the `change_me` function

Addison-Wesley is an imprint of PEARSON

Making Changes to Parameters (cont’d.)

- **Figure 3-18**
  - The `value` variable passed to the `change_me` function cannot be changed by it

Figure 3-18 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

Addison-Wesley is an imprint of PEARSON

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

Addison-Wesley
is an imprint of
PEARSON

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

Addison-Wesley
is an imprint of
PEARSON

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