GLOBAL EDITION

An Introduction to Programming Using Python™

David I. Schneider

ALWAYS LEARNING

PEARSON

An Introduction to Programming Using Python™

Global Edition

David I. Schneider

University of Maryland

PEARSON

Boston • Columbus • Indianapolis • New York • San Francisco • Hoboken

Amsterdam • Cape Town • Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto

Delhi • Mexico City • São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Vice President and Editorial Director, ECS: *Marcia J. Horton*
Executive Editor: *Tracy Johnson*
Assistant Acquisitions Editor, Global Editions: Aditee Agarwal
Executive Marketing Manager: *Tim Galligan*
Marketing Assistant: *Jon Bryant*
Team Lead Product Management: *Scott Disanno*
Production Project Manager: *Greg Dulles and Pavithra Jayapaul*
Project Editor, Global Editions: K.K. Neelakantan
Program Manager: *Carole Snyder*
Director of Operations: *Mary Fischer*
Operations Specialist: *Maura Zaldivar-Garcia*
Senior Manufacturing Controller, Global Editions: Trudy Kimber
Cover Designer: *Lumina Datamatics*
Global HE Director of Vendor Sourcing and Procurement: *Diane Hynes*
Manager, Rights and Permissions: Rachel Youdelman
Associate Project Manager, Rights and Permissions: *William Opaluch*
Media Production Manager, Global Editions: Vikram Kumar
Full-Service Project Management: *Shylaja Gattupalli, Jouve India*

Pearson Education Limited
Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the World WideWeb at:
www.pearsonglobaleditions.com

© Pearson Education Limited 2016

The right of David I. Schneider to be identified as the authors of this work has been asserted by him in accordance with the Copyright, Designs and Patents Act 1988.

Authorized adaptation from the United States edition, entitled An Introduction to Programming Using Python, ISBN 978-0-13-405822-1, by David I. Schneider published by Pearson Education © 2016.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without either the prior written permission of the publisher or a license permitting restricted copying in the United Kingdom issued by the Copyright Licensing Agency Ltd, Saffron House, 6–10 Kirby Street, London EC1N 8TS.

All trademarks used herein are the property of their respective owners. The use of any trademark in this text does not vest in the author or publisher any trademark ownership rights in such trademarks, nor does the use of such trademarks imply any affiliation with or endorsement of this book by such owners.

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

10 9 8 7 6 5 4 3 2 1

ISBN 10: 1-292-10343-4
ISBN 13: 978-1-292-10343-3

Typeset in 11/13 Goudy Old Style MT Std by Jouve India
Printed and bound by RR Donnelley Kendallville in The United States of America.

Guide to VideoNotes

www.pearsonglobaleditions.com/schneider

VideoNote

This page intentionally left blank

Guide to Application Topics

Business and Economics

Admission fee, 240
Analyze fuel economy, 257
Annual percentage yield (APY), 119
Annuity, 50, 132, 133, 148
Automobile depreciation, 148
Balance in a savings account, 136, 181, 182
Balance on a car loan, 131, 148, 155
Balance on a mortgage, 292
Bond yield-to-maturity, 91
Break-even analysis, 49
Calculate a sale price, 119, 354
Calculate a tip, 64, 72, 116
Calculate monthly payment for a mortgage, 321, 322
Calculate weekly pay, 117, 164, 309
Change from a purchase, 118
Check out purchases from a website, 310
Compare interest rates, 119
Compare salary options, 149, 179
Compare simple and compound interest, 148
Compound interest, 49, 50, 119, 124, 132, 133, 148, 165
Consumer price index (CPI), 131
Cost of electricity, 63
Credit card payment, 197
Crop production, 50, 149
Currency exchange rates, 258
Depreciation, 205
Determine a company's payroll, 326
Discounted price, 49, 62
Distribution of a mortgage payment, 197
Dogs of the DOW, 231
Doubling time of CPI, 131
DOW industrial average, 230, 231
Earnings, 197
Effects of a change in salary, 72, 73, 354, 355
Evaluate effects of different interest rates and compounding periods, 361
Excel, 227
FICA tax, 110, 179
Future value, 73, 165
Growth of an investment, 148
Income tax, 119
Individual retirement account, 156
Interest earned in a savings account, 181, 182
Interest-only mortgage, 322
Journal subscriptions, 221
Lifetime earnings, 148
Make change, 90
Manage a bank account, 133, 325
Monetary units of countries, 258
Monthly payment on a car loan, 91, 355
Mortgage with points, 322
Municipal bonds, 64
Net income, 72
Number of restaurants in United States, 50
Pay raise, 179, 197
Pension calculation, 180, 362
Percentage markup, 64
Percentage profit, 50
Pizza consumption, 50
PNC Christmas price index, 233
Postage costs, 193
Present value, 73
Price-to-earnings ratio, 63
Profit, 49, 109, 120
Profit margin, 64
Retirement plan, 156
Rule of 88, 156
Sales receipt, 236, 310
Savings plan, 132
Simple interest, 148
Small dogs of the DOW, 231
Stock portfolio, 91
Stock purchase, 49
Supply and demand, 149
Toll booth register, 310
Total cost, 116, 117
Total interest payments on a car loan, 155
U.S. national debt, 51
Unit price, 91
Validate credit card number, 157
Validate ISBN, 205
Withdrawal from a savings account, 118

General Interest

Academy award winners, 235, 357
American flag, 295
Anagrams, 194
Analyze a sentence, 87, 250
Bachelor degrees, 256
Boston accent, 151
Caffeine absorption, 156
Calculate number of calories, 51
Calculate semester grades, 311, 312, 314
Computer pioneers, 234
Country flags, 277
Crayon colors, 179, 221, 222
Determine day of week, 236

Contents

This page intentionally left blank

Preface

Since its introduction in the 1990s, Python has become one of the most widely used programming languages in the software industry. Also, students learning their first programming language find Python the ideal tool to understand the development of computer programs.

My objectives when writing this text were as follows:

1. *To develop focused chapters.* Rather than covering many topics superficially, I concentrate on important subjects and cover them thoroughly.
2. *To use examples and exercises with which students can relate, appreciate, and feel comfortable.* I frequently use real data. Examples do not have so many embellishments that students are distracted from the programming techniques illustrated.
3. *To produce compactly written text that students will find both readable and informative.* The main points of each topic are discussed first and then the peripheral details are presented as comments.
4. *To teach good programming practices that are in step with modern programming methodology.* Problem-solving techniques, structured programming, and object-oriented programming are thoroughly discussed.
5. *To provide insights into the major applications of computers.*

Unique and Distinguishing Features

Programming Projects. Beginning with Chapter 2, every chapter contains programming projects. The programming projects reflect the variety of ways that computers are used. The large number and range of difficulty of the programming projects provide the flexibility to adapt the course to the interests and abilities of the students. Some programming projects in later chapters can be assigned as end-of-the-semester projects.

Exercises for Most Sections. Each section that teaches programming has an exercise set. The exercises both reinforce the understanding of the key ideas of the section and challenge the student to explore applications. Most of the exercise sets require the student to trace programs, find errors, and write programs. The answers to every odd-numbered exercise in the book, with the exception of Section 6.3 (Turtle Graphics) and Chapter 8 (Graphical User Interface), are given at the end of the text. (The answers to every other odd-numbered exercise from Section 6.3 are given. The Student Solutions Manual contains the answer to every odd-numbered exercise in the book.) A possible output accompanies nearly every programming exercise and programming project.

Practice Problems. Practice Problems are carefully selected exercises located at the end of a section, just before the exercise set. Complete solutions are given following the exercise set. The practice problems often focus on points that are potentially confusing or are best appreciated after the student has thought about them. The reader should seriously attempt the practice problems and study their solutions before moving on to the exercises.

Comments. Extensions and fine points of new topics are deferred to the "Comments" portion at the end of each section so that they will not interfere with the flow of the presentation.

Key Terms and Concepts. In Chapters 2 through 8, the key terms and concepts (along with examples) are summarized at the end of the chapter.

Guide to Application Topics. This section provides an index of programs that deal with various topics including Business, Economics, Mathematics, and Sports.

VideoNotes. Twenty-four VideoNotes are available at www.pearsonhighered.com/schneider. VideoNotes are Pearson's visual tool designed for teaching key programming concepts and techniques. VideoNote icons are placed in the margin of the text book to notify the reader when a topic is discussed in a video. Also, a Guide to Video Notes summarizing the different videos throughout the text is included.

Solution Manuals. The Student Solutions Manual contains the answer to every odd-numbered exercise (not including programming projects). The Instructor Solutions Manual contains the answer to every exercise and programming project. Both solution manuals are in pdf format and can be downloaded from the Publisher's website.

Source Code and Data Files. The programs for all examples and the data files needed for the exercises can be downloaded from the Publisher's website.

How to Access Instructor and Student Resource Materials

Online Practice and Assessment with MyProgrammingLab™

MyProgrammingLab helps students fully grasp the logic, semantics, and syntax of programming. Through practice exercises and immediate, personalized feedback, MyProgrammingLab improves the programming competence of beginning students who often struggle with the basic concepts and paradigms of popular high-level programming languages.

A self-study and homework tool, a MyProgrammingLab course consists of hundreds of small practice problems organized around the structure of this textbook. For students, the system automatically detects errors in the logic and syntax of their code submissions and offers targeted hints that enable students to figure out what went wrong—and why. For instructors, a comprehensive gradebook tracks correct and incorrect answers and stores the code inputted by students for review.

For a full demonstration, to see feedback from instructors and students, or to get started using MyProgrammingLab in your course, visit www.myprogramminglab.com.

Instructor Resources

The following protected instructor resource materials are available on the Publisher's website at www.pearsonglobaleditions.com/schneider.

- Test Item File
- PowerPoint Lecture Slides

- Instructor Solutions Manual
- VideoNotes
- Programs for all examples and answers to exercises and programming projects (Data files needed for the exercises are included in the Programs folder.)

Student Resources

Access to the Premium website and VideoNotes tutorials is located at www.pearsonglobaleditions.com/schneider. Students must use the access card located in the front of the book to register and access the online material. Instructors must register on the site to access the material.

The following content is available through the Premium website:

- VideoNotes
- Student Solutions Manual
- Programs for examples (Data files needed for the exercises are included in the Programs folder.)

This page intentionally left blank

Acknowledgments

Many talented instructors and programmers provided helpful comments and constructive suggestions during the writing of this text and I am most grateful for their contributions. The book benefited greatly from the valuable comments of the following reviewers:

Daniel Solarek, University of Toledo
David M. Reed, Capital University
Debraj De, Georgia State
Desmond Chun, Chabot College
Mark Coffey, Colorado School of Mines
Randall Alexander, College of Charleston
Vineyak Tanksale, Ball State University
Zhi Wei, New Jersey Institute of Technology

Many people are involved in the successful publication of a book. I wish to thank the dedicated team at Pearson whose support and diligence made this textbook possible, especially Carole Snyder, Program Manager for Computer Science, Kelsey Loanes, Editorial Assistant for Computer Science, and Scott Disanno, Team Lead Product Management.

I would like to thank Jacob Saina for his assistance with every stage in the writing of the book. Production Editors Pavithra Jayapaul and Greg Dulles did a fantastic job producing the book and keeping it on schedule. I am grateful to John Russo of the Wentworth Institute of Technology for producing the VideoNotes, to Dr. Kathy Liszka of the University of Akron for producing the test bank, and to Dr. Steve Armstrong of LeTourneau University for producing the PowerPoint slides that accompany the book. The competence and graciousness of Shylaja Gattupalli at Jouve India made for a pleasant production process.

I extend special thanks to my editor Tracy Johnson. Her ideas and enthusiasm helped immensely with the preparation of the book.

David I. Schneider
dis@alum.mit.edu

Pearson would like to thank and acknowledge Shaligram Prajapat, Devi Ahilya University for contributing to the Global Edition and Somitra Sanadhya, Indian Institute of Delhi, Rosanne Els, University of Kwazulu-Natal, and Shivani Pandit for reviewing the Global Edition.

This page intentionally left blank

1

An Introduction to Computing and Problem Solving

1.1 An Introduction to Computing and Python

An Introduction to Programming Using Python is about problem solving using computers. The programming language used is Python, but the principles apply to most modern programming languages. Many of the examples and exercises illustrate how computers are used in the real world. Here are some questions that you may have about computers and programming.

Question: *How do we communicate with the computer?*

Answer: Programming languages are used to communicate with the computer. At the lowest level, there is *machine language*, which is understood directly by the microprocessor but is difficult for humans to understand. Python is an example of a *high-level language*. It consists of instructions to which people can relate, such as *print*, *if*, and *input*. Some other well-known high-level languages are Java, C++, and Visual Basic.

Question: *How do we get computers to perform complicated tasks?*

Answer: Tasks are broken down into a sequence of instructions, called a **program**, that can be expressed in a programming language. Programs can range in size from two or three instructions to millions of instructions. The process of executing the instructions is called *running* the program.

Question: *Why did you decide to use Python as the programming language?*

Answer: Many people consider Python to be the best language to teach beginners how to program. We agree. Also, Python is being used by major software companies. Python is powerful, easy to write and read, easy to download and install, and it runs under Windows, Mac, and Linux operating systems.

Question: *How did the language Python get its name?*

Answer: It is named for the British comedy group *Monty Python*. Python's creator, Guido van Rossum, is a fan of the group.

Question: *This book uses the editor IDLE to create programs. How did IDLE get its name?*

Answer: *IDLE* stands for **I**ntegrated **D**eve**L**opment **E**nvironment. (Some people think the name was chosen as a tribute to Eric Idle, a founding member of the Monty Python group.) The IDLE editor has many features (such as color coding and formatting assistance) that help the programmer.

Question: *Python is referred to as an interpreted language. What is an interpreted language?*

Answer: An interpreted language uses a program called an *interpreter* that translates a high-level language one statement at a time into machine language and then runs the program. The interpreter will spot several types of errors and terminate the program when one is encountered.

Question: *What are the meanings of the terms "programmer" and "user"?*

Answer: A *programmer* (also called a *developer*) is a person who solves problems by writing programs on a computer. After analyzing the problem and developing a plan for solving it, the programmer writes and tests the program that instructs the computer how to carry out the plan. The program might be run many times, either by the programmer or by others. A *user* is any person who runs the program. While working through this text, you will function both as a programmer and as a user.

Question: *What is the meaning of the term "code"?*

Answer: The Python instructions that the programmer writes are called *code*. The processes of writing a program is often called *coding*.

Question: *Are there certain characteristics that all programs have in common?*

Answer: Most programs do three things: take in data, manipulate data, and produce results. These operations are referred to as *input*, *processing*, and *output*. The input data might be held in the program, reside on a disk, or be provided by the user in response to requests made by the computer while the program is running. The processing of the input data occurs inside the computer and can take from a fraction of a second to many hours. The output data are displayed on a monitor, printed on a printer, or recorded on a disk. As a simple example, consider a program that computes sales tax. An item of input data is the cost of the thing purchased. The processing consists of multiplying the cost by the sales tax rate. The output data is the resulting product, the amount of sales tax to be paid.

Question: *What are the meanings of the terms "hardware" and "software"?*

Answer: *Hardware* refers to the physical components of the computer, including all peripherals, the central processing unit (CPU), disk drives, and all mechanical and electrical devices. Programs are referred to as *software*.

Question: *How are problems solved with a program?*

Answer: Problems are solved by carefully reading them to determine what data are given and what outputs are requested. Then a step-by-step procedure is devised to process the given data and produce the requested output.

Question: *Many programming languages, including Python, use a zero-based numbering system. What is a zero-based numbering system?*

Answer: In a zero-based numbering system, numbering begins with *zero* instead of *one*. For example, in the word "code", "c" would be the zeroth letter, "o" would be the first letter, and so on.

Question: *Are there any prerequisites to learning Python?*

Answer: You should be familiar with how folders (also called *directories*) and files are managed on your computer. Files reside on storage devices such as hard disks, USB flash drives, CDs, and DVDs. Traditionally, the primary storage devices for personal computers were hard disks and floppy disks. Therefore, the word *disk* is frequently used to refer to any storage device.

Question: *What is an example of a program developed in this textbook?*

Answer: Figure 1.1 shows a possible output of a program from Chapter 3. When it is first run, the statement "Enter a first name:" appears. After the user types in a first name and

```
Enter a first name: James
James Madison
James Monroe
James Polk
James Buchanan
James Garfield
James Carter
```

FIGURE 1.1 **A possible output for a program in Chapter 3.**

presses the *Enter* (or *return*) key, the names of the presidents who have that first name are displayed.

Question: *How does the programmer create the aforementioned program?*

Answer: For this program, the programmer writes about 10 lines of code that search a text file named **`USpres.txt`**, and extracts the requested names.

Question: *What conventions are used to show keystrokes?*

Answer: The combination *key1+key2* means "hold down key1 and then press key2". The combination Ctrl+C places selected material into the Clipboard. The combination *key1/key2* means "press and release key1, and then press key2". The combination Alt/F opens the *File* menu on a menu bar.

Question: *How can the programs for the examples in this textbook be obtained?*

Answer: See the preface for information on how to download the programs from the Pearson website.

Question: *Where will new programs be saved?*

Answer: Before writing your first program, you should create a special folder to hold your programs.

1.2 Program Development Cycle

We learned in Section 1.1 that hardware refers to the machinery in a computer system (such as the monitor, keyboard, and CPU) and software refers to a collection of instructions, called a **program**, that directs the hardware. Programs are written to solve problems or perform tasks on a computer. Programmers translate the solutions or tasks into a language the computer can understand. As we write programs, we must keep in mind that the computer will do only what we instruct it to do. Because of this, we must be very careful and thorough when writing our instructions.

Performing a Task on the Computer

The first step in writing instructions to carry out a task is to determine what the **output** should be—that is, exactly what the task should produce. The second step is to identify the data, or **input**, necessary to obtain the output. The last step is to determine how to **process** the input to obtain the desired output—that is, to determine what formulas or ways of doing things should be used to obtain the output.

This problem-solving approach is the same as that used to solve word problems in an algebra class. For example, consider the following algebra problem:

How fast is a car moving if it travels 50 miles in 2 hours?

The first step is to determine the type of answer requested. The answer should be a number giving the speed in miles per hour (the output). The information needed to obtain the answer is the distance and time the car has traveled (the input). The formula

$$\text{speed} = \text{distance/time}$$

is used to process the distance traveled and the time elapsed in order to determine the speed. That is,

$$\begin{aligned} \text{speed} &= 50 \text{ miles/2 hours} \\ &= 25 \text{ miles per hour} \end{aligned}$$

A graphical representation of this problem-solving process is shown in Fig. 1.2.

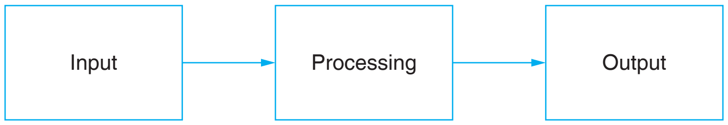


FIGURE 1.2 **The problem-solving process.**

We determine what we want as output, get the needed input, and process the input to produce the desired output.

In the chapters that follow, we discuss how to write programs to carry out the preceding operations. But first we look at the general process of writing programs.

Program Planning

A baking recipe provides a good example of a plan. The ingredients and the amounts are determined by what is to be baked. That is, the *output* determines the *input* and the *processing*. The recipe, or plan, reduces the number of mistakes you might make if you tried to bake with no plan at all. Although it's difficult to imagine an architect building a bridge or a factory without a detailed plan, many programmers (particularly students in their first programming course) try to write programs without first making a careful plan. The more complicated the problem, the more complex the plan must be. You will spend much less time working on a program if you devise a carefully thought out step-by-step plan and test it before actually writing the program.

Many programmers plan their programs using a sequence of steps, referred to as the **Software Development Life Cycle**. The following step-by-step process will enable you to use your time efficiently and help you design error-free programs that produce the desired output.

1. ***Analyze:*** Define the problem.

   Be sure you understand what the program should do—that is, what the output should be. Have a clear idea of what data (or input) are given and the relationship between the input and the desired output.

2. ***Design:*** Plan the solution to the problem.

   Find a logical sequence of precise steps that solve the problem. Such a sequence of steps is called an **algorithm**. Every detail, including obvious steps, should appear in the algorithm. In the next section, we discuss three popular methods used to develop the logic plan: flowcharts, pseudocode, and hierarchy charts. These tools help the programmer break a problem into a sequence of small tasks the computer can perform to solve the problem. Planning also involves using representative data to test the logic of the algorithm by hand to ensure that it is correct.

3. ***Code:*** Translate the algorithm into a programming language.

   Coding is the technical word for writing the program. During this stage, the program is written in Python and entered into the computer. The programmer uses the algorithm devised in Step 2 along with a knowledge of Python.

4. ***Test and correct:*** Locate and remove any errors in the program.

   Testing is the process of finding errors in a program. (An error in a program is called a **bug** and testing and correcting is often referred to as **debugging**.) As the program is typed, Python points out certain kinds of program errors. Other kinds of errors are detected by Python when the program is executed—however, many errors due to typing mistakes, flaws in the algorithm, or incorrect use of the Python language rules, can be uncovered and corrected only by careful detective work. An example of such an error would be using addition when multiplication was the proper operation.

5. ***Complete the documentation:*** Organize all the material that describes the program.

   Documentation is intended to allow another person, or the programmer at a later date, to understand the program. Internal documentation (comments) consists of statements in the program that are not executed, but point out the purposes of various parts of the program. Documentation might also consist of a detailed description of what the program does and how to use it (for instance, what type of input is expected). For commercial programs, documentation includes an instruction manual and on-line help. Other types of documentation are the flowchart, pseudocode, and hierarchy chart that were used to construct the program. Although documentation is listed as the last step in the program development cycle, it should take place as the program is being coded.

1.3 Programming Tools

This section discusses some specific algorithms and describes three tools used to convert algorithms into computer programs: flowcharts, pseudocode, and hierarchy charts.

You use algorithms every day to make decisions and perform tasks. For instance, whenever you mail a letter, you must decide how much postage to put on the envelope. One rule of thumb is to use one stamp for every five sheets of paper or fraction thereof. Suppose a friend asks you to determine the number of stamps to place on an envelope. The following algorithm will accomplish the task.

1. Request the number of sheets of paper; call it Sheets. (*input*)
2. Divide Sheets by 5. (*processing*)
3. If necessary, round the quotient up to a whole number; call it Stamps. (*processing*)
4. Reply with the number Stamps. (*output*)

The preceding algorithm takes the number of sheets (Sheets) as input, processes the data, and produces the number of stamps needed (Stamps) as output. We can test the algorithm for a letter with 16 sheets of paper.

1. Request the number of sheets of paper; Sheets = 16.
2. Dividing 5 into 16 gives 3.2.
3. Rounding 3.2 up to 4 gives Stamps = 4.
4. Reply with the answer, 4 stamps.

This problem-solving example can be illustrated by Fig. 1.3.

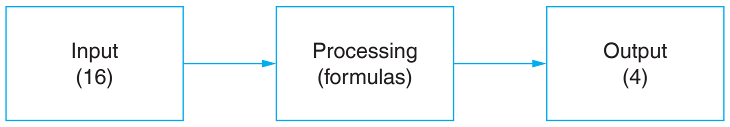


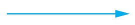
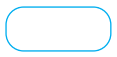
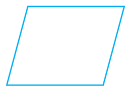
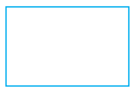
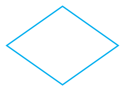
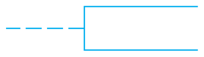
FIGURE 1.3 **The problem-solving process for the stamp problem.**

Of the program design tools available, three popular ones are the following:

Flowcharts: Graphically depict the logical steps to carry out a task and show how the steps relate to each other.
Pseudocode: Uses English-like phrases with some Python terms to outline the task.
Hierarchy charts: Show how the different parts of a program relate to each other.

Flowcharts

A flowchart consists of special geometric symbols connected by arrows. Within each symbol is a phrase presenting the activity at that step. The shape of the symbol indicates the type of operation that is to occur. For instance, the parallelogram denotes input or output. The arrows connecting the symbols, called **flowlines**, show the progression in which the steps take place. Flowcharts should "flow" from the top of the page to the bottom. Although the symbols used in flowcharts are standardized, no standards exist for the amount of detail required within each symbol.

Symbol	Name	Meaning
	***Flowline***	Used to connect symbols and indicate the flow of logic.
	***Terminal***	Used to represent the beginning (Start) or the end (End) of a task.
	***Input/Output***	Used for input and output operations. The data to be input or output is described in the parallelogram.
	***Processing***	Used for arithmetic and data-manipulation operations. The instructions are listed inside the rectangle.
	***Decision***	Used for any logic or comparison operations. Unlike the input/output and processing symbols, which have one entry and one exit flowline, the decision symbol has one entry and two exit paths. The path chosen depends on whether the answer to a question is "yes" or "no."
	Connector	Used to join different flowlines.
	***Annotation***	Used to provide additional information about another flowchart symbol.

The table of the flowchart symbols has been adopted by the American National Standards Institute (ANSI). Figure 1.4 shows the flowchart for the postage-stamp problem.

The main advantage of using a flowchart to plan a task is that it provides a graphical representation of the task, thereby making the logic easier to follow. We can clearly see every step and how each is connected to the next. The major disadvantage is that when a program is very large, the flowcharts may continue for many pages, making them difficult to follow and modify.

Pseudocode

Pseudocode is an abbreviated plain English version of actual computer code (hence, *pseudocode*). The geometric symbols used in flowcharts are replaced by English-like statements that outline the process. As a result, pseudocode looks more like computer code than does

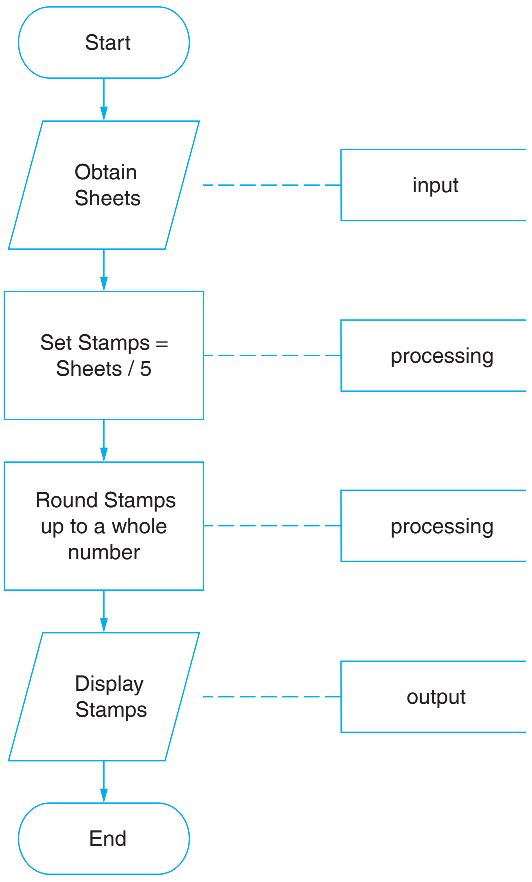


FIGURE 1.4 **Flowchart for the postage-stamp problem.**

a flowchart. Pseudocode allows the programmer to focus on the steps required to solve a problem rather than on how to use the computer language. The programmer can describe the algorithm in Python-like form without being restricted by the rules of Python. When the pseudocode is completed, it can be easily translated into the Python language.

The pseudocode for the postage-stamp problem is shown in Fig. 1.5.

```
Program: Determine the proper number of stamps for a letter.
Obtain number of sheets (Sheets)                              (input)
Set the number of stamps to Sheets / 5                   (processing)
Round the number of stamps up to a whole number          (processing)
Display the number of stamps                                 (output)
```

FIGURE 1.5 **Pseudocode for the postage-stamp problem.**

Pseudocode has several advantages. It is compact and probably will not extend for many pages as flowcharts commonly do. Also, the pseudocode looks like the code to be written and so is preferred by many programmers.

Hierarchy Chart

The last programming tool we'll discuss is the **hierarchy chart**, which shows the overall program structure. Hierarchy charts are also called structure charts, HIPO (Hierarchy plus Input-Process-Output) charts, top-down charts, or VTOC (Visual Table of Contents) charts. All these names refer to planning diagrams that are similar to a company's organization chart.

Hierarchy charts depict the organization of a program but omit the specific processing logic. They describe what each part of the program does and they show how the parts relate to each other. The details on how the parts work, however, are omitted. The chart is read from top to bottom and from left to right. Each part may be subdivided into a succession of subparts that branch out under it. Typically, after the activities in the succession of subparts are carried out, the part to the right of the original part is considered. A quick glance at the hierarchy chart reveals each task performed in the program and where it is performed. Figure 1.6 shows a hierarchy chart for the postage-stamp problem.

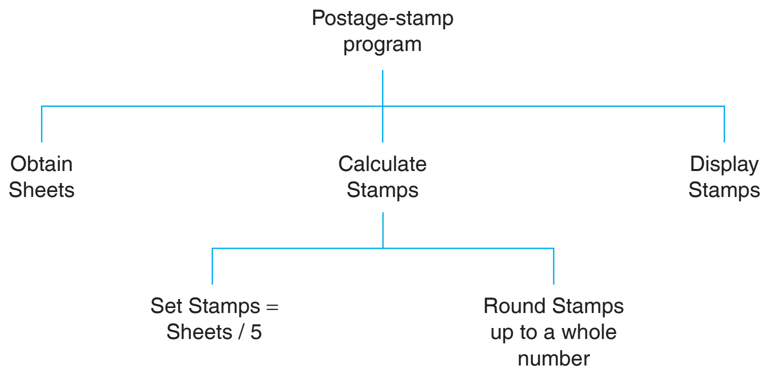


FIGURE 1.6 **Hierarchy chart for the postage-stamp problem.**

The main benefit of hierarchy charts is in the initial planning of a program. We break down the major parts of a program so we can see what must be done in general. From this point, we can then refine each part into more detailed plans using flowcharts or pseudocode. This process is called the **divide-and-conquer** method.

Decision Structure

The postage-stamp problem was solved by a series of instructions to obtain the data, perform calculations, and display the results. Each step was in a sequence, that is, we moved from one line to the next without skipping over any lines. This kind of structure is called a **sequence structure**. Many problems, however, require a decision to determine whether a series of instructions should be executed. If the answer to a question is "yes", then one group of instructions is executed. If the answer is "no", then another is executed. This structure is called a **decision structure**. Figure 1.7 contains the pseudocode and flowchart for a decision structure.

Sequence and decision structures are both used to solve the following problem.

```
If condition is true
    Process step(s) 1
else
    Process step(s) 2
```

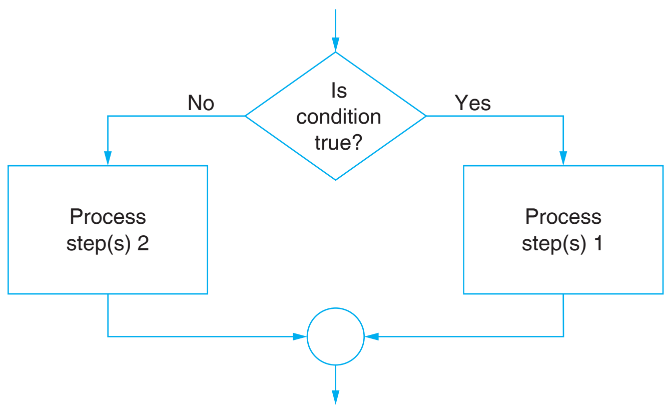


FIGURE 1.7 **Pseudocode and flowchart for a decision structure.**

■ Direction of Numbered NYC Streets Algorithm

Problem: Given a street number of a one-way street in New York City, decide the direction of the street, either eastbound or westbound.

Discussion: There is a simple rule to tell the direction of a one-way street in New York City: Even-numbered streets run eastbound.

Input: Street number.

Processing: Decide if the street number is divisible by 2.

Output: "Eastbound" or "Westbound".

Figures 1.8 through 1.10 show the flowchart, pseudocode, and hierarchy chart for the numbered New York City streets problem.

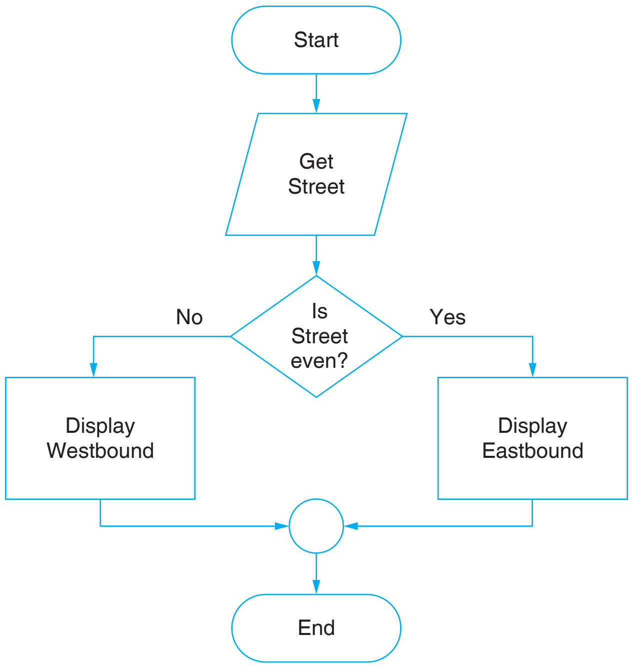


FIGURE 1.8 **Flowchart for the numbered New York City streets problem.**

Program: Determine the direction of a numbered NYC street.

```
Get street
if street is even
   Display Eastbound
else
   Display Westbound
```

FIGURE 1.9 **Pseudocode for the numbered New York City streets problem.**

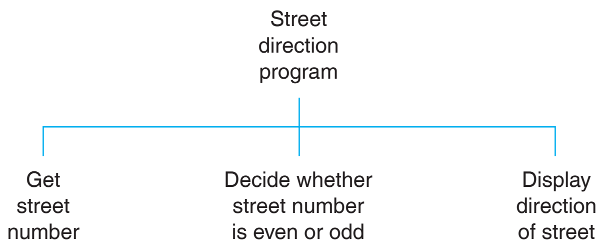

Street direction program
Get street number
Decide whether street number is even or odd
Display direction of street

FIGURE 1.10 **Hierarchy chart for the numbered New York City streets problem.**

Repetition Structure

A programming structure that executes instructions many times is called a **repetition structure** or a **loop structure**. Loop structures need a test (or condition) to tell when the loop should end. Without an exit condition, the loop would repeat endlessly (an infinite loop). One way to control the number of times a loop repeats (often referred to as the number of passes or iterations) is to check a condition before each pass through the loop and continue executing the loop as long as the condition is true. See Fig. 1.11. The solution of the next problem requires a repetition structure.

```
Do While condition is true
    Process step(s)
Loop
```

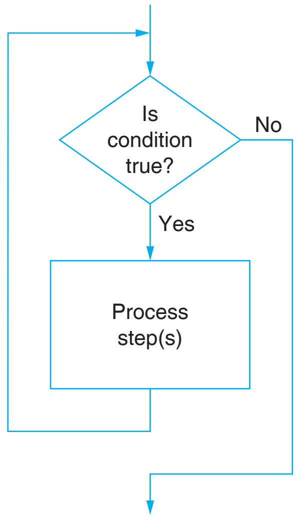


FIGURE 1.11 **Pseudocode and flowchart for a loop.**

Class Average Algorithm

Problem: Calculate and report the average grade for a class.

Discussion: The average grade equals the sum of all grades divided by the number of students. We need a loop to get and then add (accumulate) the grades for each student in the class. Inside the loop, we also need to total (count) the number of students in the class. See Figs. 1.12 to 1.14 on the next page.

Input: Student grades.

Processing: Find the sum of the grades; count the number of students; calculate average grade sum of grades / number of students.

Output: Average grade.

Comments

1. Tracing a flowchart is like playing a board game. We begin at the Start symbol and proceed from symbol to symbol until we reach the End symbol. At any time, we will be at just one symbol. In a board game, the path taken depends on the result of spinning a spinner or rolling a pair of dice. The path taken through a flowchart depends on the input.
2. The algorithm should be tested at the flowchart stage before being coded into a program. Different data should be used as input, and the output checked. This process is known as **desk checking**. The test data should include nonstandard data as well as typical data.

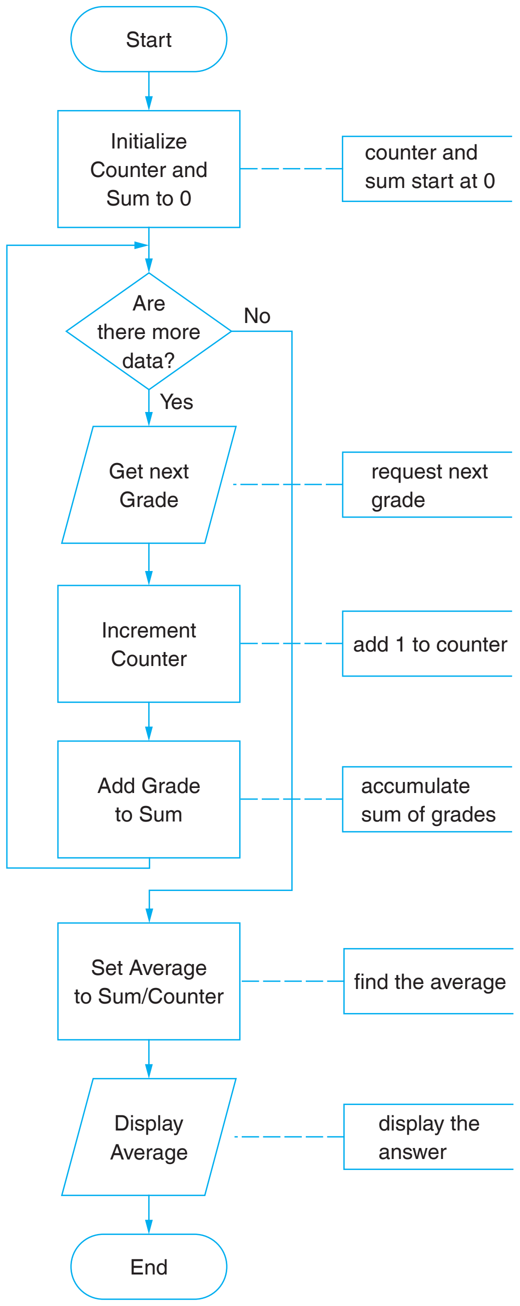


FIGURE 1.12 **Flowchart for the class average problem.**

```
Program: Calculate and report the average grade of a class.
Initialize Counter and Sum to 0
while there are more data
    Get the next Grade
    Increment the Counter
    Add the Grade to the Sum
Set Average to Sum / Counter
Display Average
```

FIGURE 1.13 **Pseudocode for the class average problem.**

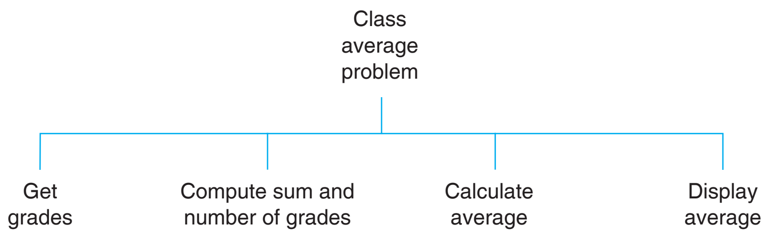


FIGURE 1.14 **Hierarchy chart for the class average problem.**

3. Flowcharts, pseudocode, and hierarchy charts are universal problem-solving tools. They can be used to plan programs for implementation in many computer languages, not just Python.
4. Flowcharts are time-consuming to write and difficult to update. For this reason, professional programmers are more likely to favor pseudocode and hierarchy charts. Because flowcharts so clearly illustrate the logical flow of programming techniques, they are a valuable tool in the education of programmers.
5. There are many styles of pseudocode. Some programmers use an outline form, whereas others use a form that looks almost like a programming language. Several Python keywords, such as "if", "else", and "while", are used extensively in pseudocode.

1.4 An Introduction to Python

The discussions in this book refer to IDLE, the editor that ships with Python. You should be able to carry out the tasks from the book with a different editor by making simple adjustments. We will assume that Python 3 is installed on your computer along with IDLE (or whatever editor you have decided to use). If necessary, see Appendix C for instructions on installing Python and IDLE.

■ Starting IDLE

WINDOWS: Depending on the version of Windows you are using, you should be able to invoke IDLE with a sequence like *Start/All Programs/Python 34/IDLE* or by clicking on a tile similar to the one in Fig. 1.15.

FIGURE 1.15 **IDLE tile from Windows.**

MAC: To invoke IDLE, open Finder, select *Applications*, select the *Python 3.x* folder, and run IDLE from there.
LINUX and UNIX: To invoke IDLE, run *idle3* from the folder */usr/bin*. IDLE can also be invoked by entering *idle3* into a terminal.