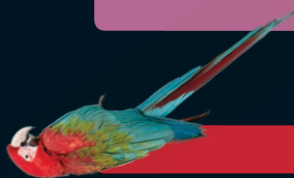


# CAY HORSTMANN RANCE D. NECAISE



# PYTHON

FOR EVERYONE



## Variable and Constant Definitions

```
      Name      Initial value
    /         /
cansPerPack = 6
CAN_VOLUME = 0.335
```

Use uppercase for constants

## Mathematical Functions

```
abs(x)           Absolute value |x|
round(x)         Rounds to nearest integer
max(x1, x2, ...) Largest of the arguments
min(x1, x2, ...) Smallest of the arguments
```

From math module:

```
sqrt(x)          Square root  $\sqrt{x}$ 
trunc(x)         Truncates to an integer
sin(x), cos(x), tan(x) Sine, cosine, tangent of x
degrees(x), radians(x) Converts to degrees or radians
log(x), log(x, base) Natural log,  $\log_{\text{base}}(x)$ 
```

## Imports

```
      Module      Imported items
    /             /
from math import sqrt, log
```

## Conditional Statement

```
      Condition
    /
if floor >= 13 : Executed when condition is true
    actualFloor = floor - 1
elif floor >= 0 : Second condition (optional)
    actualFloor = floor
else : Executed when all conditions are false (optional)
    print("Floor negative")
```

## Loop Statements

```
      Condition
    /
while balance < TARGET : Executed while condition is true
    year = year + 1
    balance = balance * (1 + rate / 100)
]

A container (list, str, range, dict, set)
for value in values :
    sum = sum + value
```

## Function Definition

```
      Function name      Parameter name
    /                   /
def cubeVolume(sideLength) :
    volume = sideLength ** 3
    return volume

Exits method and returns result
```

## Selected Operators and Their Precedence

(See Appendix B for the complete list.)

```
[]           Sequence element access
**          Raising to a power
* / // %    Multiplication, division, floor
            division, remainder
+ -         Addition, subtraction
< <= > >= != in Comparisons and membership
not         } Boolean operators
or          }
and         }
```

## Strings

```
s = "Hello"
len(s)       The length of the string: 5
s[1]        The character with index 1: "e"
s + "!"     Concatenation: Hello!
s * 2       Replication: "HelloHello"
s.upper()   Yields "HELLO"
s.replace("e", "3") Yields "H3llo"
```

## Lists

```
friends = [] — An empty list
values = [16, 3, 2, 13]
```

```
for i in range(len(values))
    values[i] = i * i
```

```
friends.append("Bob")
friends.insert(0, "Amy")
if "Amy" in friends :
    n = friends.index("Amy")
    friends.pop(n) — Removes nth
else :
    friends.pop() — Removes last
friends.remove("Bob")
```

```
guests = friends + ["Lee", "Zoe"] — Concatenation
scores = [0] * 12 — Replication
bestFriends = friends[0 : 3] — Slice
                        Included Excluded
```

```
total = sum(values) — List must contain numbers
largest = max(values)
values.sort() — Use min to get the smallest
```

## Tables

```
table = [[16, 3, 2, 13],
         [5, 10, 11, 8],
         [9, 6, 7, 12],
         [4, 15, 14, 1]]
```

Number of rows

Number of columns

```
for row in range(len(table)) :
    for column in range(len(table[row])) :
        sum = sum + table[row][column]
```

# PYTHON FOR EVERYONE



**Cay Horstmann**

San Jose State University

**Rance D. Necaie**

College of William and Mary

**WILEY**

VP AND EXECUTIVE PUBLISHER  
EXECUTIVE EDITOR  
EDITORIAL ASSISTANT  
EXECUTIVE MARKETING MANAGER  
PRODUCTION MANAGEMENT SERVICES  
CREATIVE DIRECTOR  
SENIOR DESIGNER  
PHOTO MANAGER  
SENIOR PHOTO EDITOR  
COVER CREDIT

Don Fowley  
Beth Lang Golub  
Katherine Willis  
Christopher Ruel  
Cindy Johnson  
Harry Nolan  
Madelyn Lesure  
Hilary Newman  
Lisa Gee  
Based on Monty Python and the Holy Grail.  
Reproduced with permission of Python (Monty)  
Pictures, Ltd.  
(balloon) © Mikhail Mishchenko/123RF.com;  
(binoculars) © Tom Horyn/iStockphoto;  
(castle) © Anik Messier/Getty Images;  
(cow) © Eric Isselée/Shutterstock;  
(parrot) © Eric Isselée/iStockphoto;  
(trebuchet) © Stephen Coburn/123RF.com;  
(trumpets) © modella/123RF.com.

COVER PHOTOS

This book was set in Stempel Garamond by Publishing Services, and printed and bound by Quad/Graphics. The cover was printed by Quad/Graphics.

This book is printed on acid-free paper. ∞

Founded in 1807, John Wiley & Sons, Inc. has been a valued source of knowledge and understanding for more than 200 years, helping people around the world meet their needs and fulfill their aspirations. Our company is built on a foundation of principles that include responsibility to the communities we serve and where we live and work. In 2008, we launched a Corporate Citizenship Initiative, a global effort to address the environmental, social, economic, and ethical challenges we face in our business. Among the issues we are addressing are carbon impact, paper specifications and procurement, ethical conduct within our business and among our vendors, and community and charitable support. For more information, please visit our website: [www.wiley.com/go/citizenship](http://www.wiley.com/go/citizenship).

Copyright © 2014 John Wiley & Sons, Inc. 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, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, website [www.copyright.com](http://www.copyright.com). Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030-5774, (201) 748-6011, fax (201) 748-6008, website [www.wiley.com/go/permissions](http://www.wiley.com/go/permissions).

Evaluation copies are provided to qualified academics and professionals for review purposes only, for use in their courses during the next academic year. These copies are licensed and may not be sold or transferred to a third party. Upon completion of the review period, please return the evaluation copy to Wiley. Return instructions and a free of charge return shipping label are available at [www.wiley.com/go/returnlabel](http://www.wiley.com/go/returnlabel). Outside of the United States, please contact your local representative.

ISBN 978-1-118-62613-9  
ISBN 978-1-118-64520-8 (BRV)

Printed in the United States of America

10987654321

*For Clora, maybe—C.H.*

*To my parents  
Willard and Ella—R.N.*



# PREFACE

This book is an introduction to computer programming using Python that focuses on the essentials—and on effective learning. Designed to serve a wide range of student interests and abilities, it is suitable for a first course in programming for computer scientists, engineers, and students in other disciplines. No prior programming experience is required, and only a modest amount of high school algebra is needed. For pedagogical reasons, the book uses Python 3, which is more regular than Python 2.

Here are the book’s key features:

## **Present fundamentals first.**

The book takes a traditional route, first stressing control structures, functions, procedural decomposition, and the built-in data structures. Objects are used when appropriate in the early chapters. Students start designing and implementing their own classes in Chapter 9.

## **Guidance and worked examples help students succeed.**

Beginning programmers often ask “How do I start? Now what do I do?” Of course, an activity as complex as programming cannot be reduced to cookbook-style instructions. However, step-by-step guidance is immensely helpful for building confidence and providing an outline for the task at hand. “Problem Solving” sections stress the importance of design and planning. “How To” guides help students with common programming tasks. Numerous worked examples demonstrate how to apply chapter concepts to interesting problems.

## **Practice makes perfect.**

Of course, programming students need to be able to implement nontrivial programs, but they first need to have the confidence that they can succeed. This book contains a substantial number of self-check questions at the end of each section. “Practice It” pointers suggest exercises to try after each section. A bank of quiz and test questions covering computer science concepts and programming skills is available to instructors.

## **A visual approach motivates the reader and eases navigation.**

Photographs present visual analogies that explain the nature and behavior of computer concepts. Step-by-step figures illustrate complex program operations. Syntax boxes and example tables present a variety of typical and special cases in a compact format. It is easy to get the “lay of the land” by browsing the visuals, before focusing on the textual material.



*Visual features help the reader with navigation.*

## **Focus on the essentials while being technically accurate.**

An encyclopedic coverage is not helpful for a beginning programmer, but neither is the opposite—reducing the material to a list of simplistic bullet points. In this book, the essentials are presented in digestible chunks, with separate notes that go deeper into good practices or language features when the reader is ready for the additional information.

# A Tour of the Book

Figure 1 shows the dependencies between the chapters and how topics are organized. The core material of the book is:

- Chapter 1. Introduction
- Chapter 2. Programming with Numbers and Strings
- Chapter 3. Decisions
- Chapter 4. Loops
- Chapter 5. Functions
- Chapter 6. Lists
- Chapter 7. Files and Exceptions
- Chapter 8. Sets and Dictionaries

Two chapters cover object-oriented programming:

- Chapter 9. Objects and Classes
- Chapter 10. Inheritance

Two chapters support a course that goes more deeply into algorithm design and analysis:

- Chapter 11. Recursion
- Chapter 12. Sorting and Searching

Any chapters can be incorporated into a custom print version of this text; ask your Wiley sales representative for details.

**Appendices** Five appendices provide a handy reference for students:

- Appendix A. The Basic Latin and Latin-1 Subsets of Unicode
- Appendix B. Python Operator Summary
- Appendix C. Python Reserved Word Summary
- Appendix D. The Python Library
- Appendix E. Binary Numbers and Bit Operations

## Problem Solving Strategies

Throughout the book, students will find practical, step-by-step guidance to help them devise and evaluate solutions to programming problems. Introduced where they are most relevant, these strategies address barriers to success for many students. Strategies included are:

- Algorithm Design (with pseudocode)
- First Do It By Hand (doing sample calculations by hand)
- Flowcharts
- Test Cases
- Hand-Tracing
- Storyboards
- Reusable Functions
- Stepwise Refinement
- Adapting Algorithms
- Discovering Algorithms by Manipulating Physical Objects
- Tracing Objects (identifying state and behavior)
- Patterns for Object Data
- Thinking Recursively
- Estimating the Running Time of an Algorithm



## An Optional Graphics Library

Writing programs that create drawings can provide students with effective visualizations of complex topics. We provide a very simple graphics library that we introduce in Chapter 2. Subsequent chapters contain worked examples and exercises that use the library. This material is completely optional.

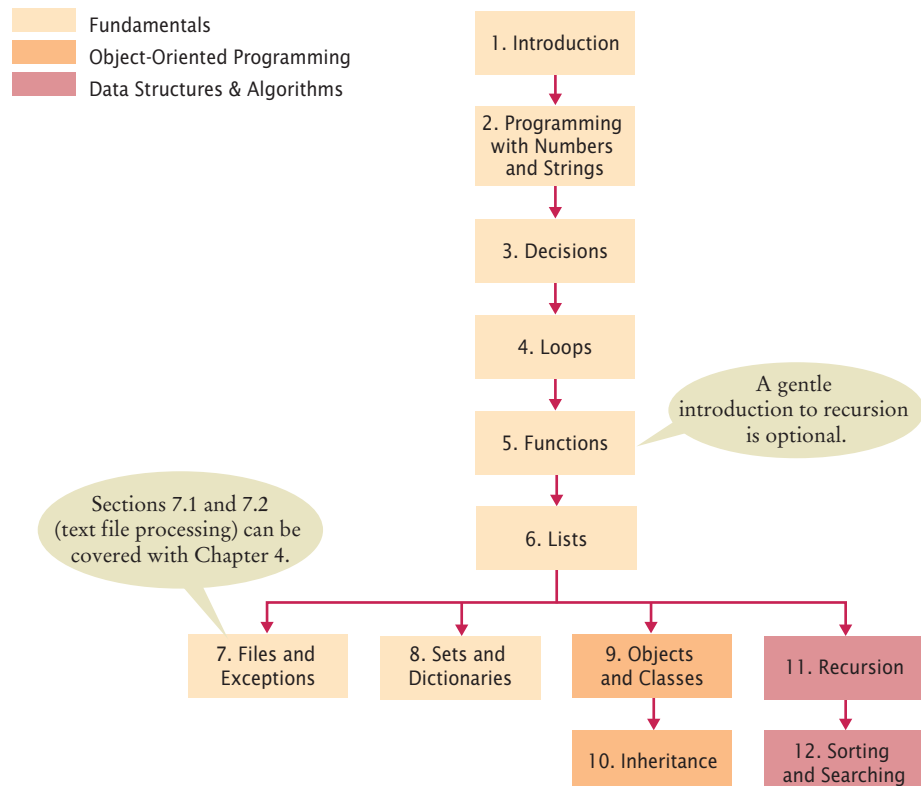
## Exercises

End-of-chapter exercises contain a broad mix of review and programming questions, with optional questions from the domains of graphics, science, and business. Designed to engage students, the exercises illustrate the value of programming in applied fields.

## Web Resources

This book is complemented by a complete suite of online resources. Go to [www.wiley.com/college/horstmann](http://www.wiley.com/college/horstmann) to visit the online companion sites, which include

- Source code for all examples in the book.
- Lecture presentation slides (in PowerPoint format).
- Solutions to all review and programming exercises (for instructors only).
- A test bank that focuses on skills, not just terminology (for instructors only).



**Figure 1**  
Chapter  
Dependencies

# A Walkthrough of the Learning Aids

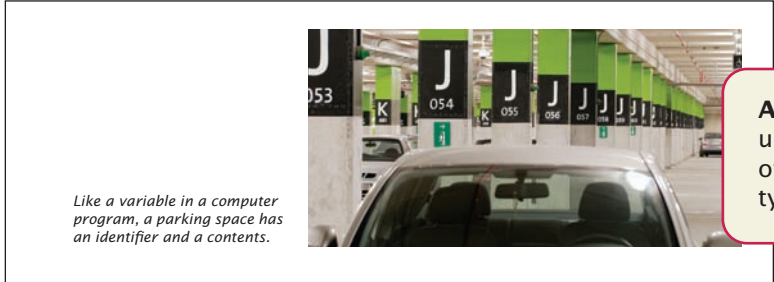
The pedagogical elements in this book work together to focus on and reinforce key concepts and fundamental principles of programming, with additional tips and detail organized to support and deepen these fundamentals. In addition to traditional features, such as chapter objectives and a wealth of exercises, each chapter contains elements geared to today’s visual learner.

Throughout each chapter, **margin notes** show where new concepts are introduced and provide an outline of key ideas.

Annotated **syntax boxes** provide a quick, visual overview of new language constructs.

**Annotations** explain required components and point to more information on common errors or best practices associated with the syntax.


**Analogies** to everyday objects are used to explain the nature and behavior of concepts such as variables, data types, loops, and more.



Like a variable in a computer program, a parking space has an identifier and a contents.

1.5 Analyzing Your First Program 11

## 1.5 Analyzing Your First Program



A comment provides information to the programmer.

A function is a collection of instructions that perform a particular task.

A function is called by specifying the function name and its arguments.

In this section, we will analyze the first Python program in detail. Here again is the code:

```

ch01/hello.py
1 # My first Python program.
2 print("Hello, World!")

```

A Python program contains one or more lines of instructions or **statements** that will be translated and executed by the Python interpreter. The first line `# My first Python program.` is a **comment**. Comments begin with `#` and are not statements. They provide descriptive information to the programmer. Comments will be discussed in more detail in Section 2.1.5.

The second line contains a statement `print("Hello, World!")` that prints or displays a line of text, namely "Hello, World!". In this statement, we call a function named `print` and pass it the information to be displayed. A **function** is a collection of programming instructions that carry out a particular task. We do not have to implement this function, it is part of the Python language. We simply want the function to perform its intended task, namely to print a value.

To use, or call, a function in Python, you need to specify

1. The name of the function you want to use (in this case, `print`).
2. Any values the function needs to carry out its task (in this case, `"Hello, World!"`). The technical term for such a value is an **argument**. Arguments are enclosed in parentheses with multiple arguments separated by commas. The number of arguments required depends on the function.

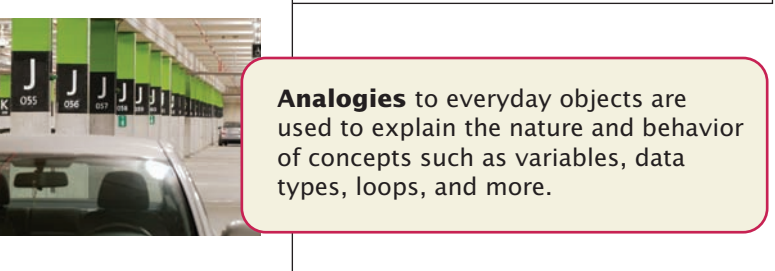
Syntax 1.1 print Statement

**Syntax** `print()`  
`print(value1, value2, ..., valuen)`

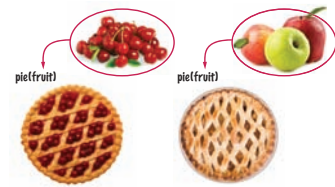
All arguments are optional. If no arguments are given, a blank line is printed.

`print("The answer is", 6 + 7, "!")`

The values to be printed, one after the other, separated by a blank space.



**Memorable photos** reinforce analogies and help students remember the concepts.



A recipe for a fruit pie may say to use any kind of fruit. Here, "fruit" is an example of a parameter variable. Apples and cherries are examples of arguments.

**Problem Solving sections** teach techniques for generating ideas and evaluating proposed solutions, often using pencil and paper or other artifacts. These sections emphasize that most of the planning and problem solving that makes students successful happens away from the computer.

6.6 Problem Solving: Discovering Algorithms by Manipulating Physical Objects 311

Now how does that help us with our problem, switching the first and the second half of the list? Let's put the first coin into place, by swapping it with the fifth coin. However, as Python programmers, we will say that we swap the coins in positions 0 and 4:



**HOW TO 1.1** Describing an Algorithm with Pseudocode

This is the first of many "How To" sections in this book that give you step-by-step procedures for carrying out important tasks in developing computer programs.

Before you are ready to write a program in Python, you need to develop an algorithm—a method for arriving at a solution for a particular problem. Describe the algorithm in pseudocode: a sequence of precise steps formulated in English.

**Problem Statement** You have the choice of buying two cars. One is more fuel efficient than the other, but also more expensive. You know the price and fuel efficiency (in miles per gallon, mpg) of both cars. You plan to keep the car for ten years. Assume a price of \$4 per gallon of gas and usage of 15,000 miles per year. You will pay cash for the car and not worry about financing costs. Which car is the better deal?



- Step 1** Determine the inputs and outputs.
- In our sample problem, we have these inputs:
- purchase price 1 and fuel efficiency 1, the price and fuel efficiency (in mpg) of the first car

**How To guides** give step-by-step guidance for common programming tasks, emphasizing planning and testing. They answer the beginner's question, "Now what do I do?" and integrate key concepts into a problem-solving sequence.

**WORKED EXAMPLE 1.1** Writing an Algorithm for Tiling a Floor

**Problem Statement** Make a plan for tiling a rectangular bathroom floor with alternating black and white tiles measuring 4 × 4 inches. The floor dimensions, measured in inches, are multiples of 4.

**Step 1** Determine the inputs and outputs.

The inputs are the floor dimensions (length × width), measured in inches. The output is a tiled floor.

**Step 2** Break down the problem into smaller tasks.

A natural subtask is to lay one row of tiles. If you can solve that task, then you can solve the problem by laying one row next to the other, starting from a wall, until you reach the opposite wall.

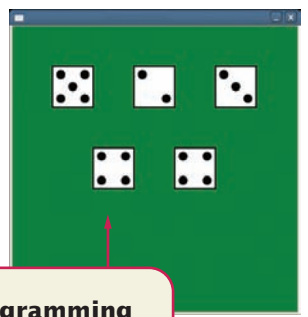


**Worked Examples** apply the steps in the How To to a different example, showing how they can be used to plan, implement, and test a solution to another programming problem.

**Table 1** Number Literals in Python

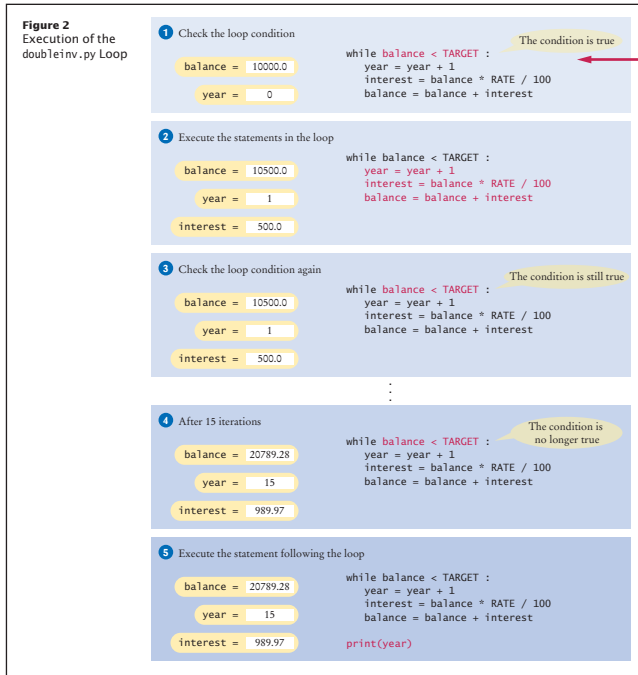
Number	Type	Comment
6	int	An integer has no fractional part.
-6	int	Integers can be negative.
0	int	Zero is an integer.
0.5	float	A number with a fractional part has type float.
1.0	float	An integer with a fractional part .0 has type float.
1E6	float	A number in exponential notation: $1 \times 10^6$ or 1000000. Numbers in exponential notation always have type float.
2.96E-2	float	Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$
100,000		<b>Error:</b> Do not use a comma as a decimal separator.
3 1/2		<b>Error:</b> Do not use fractions; use decimal notation: 3.5.

**Example tables** support beginners with multiple, concrete examples. These tables point out common errors and present another quick reference to the section's topic.

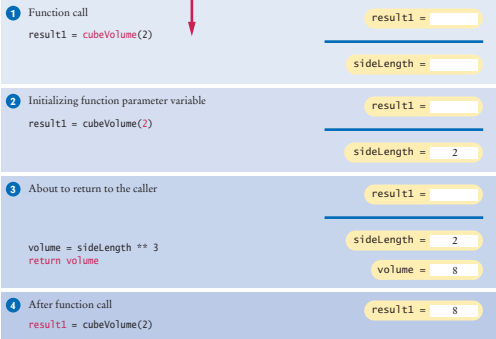


Optional **graphics programming examples** demonstrate constructs with engaging drawings, visually reinforcing programming concepts.

x Walkthrough



**Progressive figures** trace code segments to help students visualize the program flow. Color is used consistently to make variables and other elements easily recognizable.



Consider the function call illustrated in Figure 3:  
`result1 = cubeVolume(2)`

- The parameter variable `sideLength` of the `cubeVolume` function is created when the function is called. 1
- The parameter variable is initialized with the value of the argument that was passed in the call. In our case, `sideLength` is set to 2. 2
- The function computes the expression `sideLength ** 3`, which has the value 8. That value is stored in the variable `volume`. 3
- The function returns. All of its variables are removed. The return value is transferred

**Self-check exercises** at the end of each section are designed to make students think through the new material—and can spark discussion in lecture.



- SELF-CHECK**
- Write the for loop of the `investment.py` program as a `while` loop.
  - How many numbers does this loop print?  
`for n in range(10, -1, -1):`  
`print(n)`
  - Write a for loop that prints all even numbers between 10 and 20 (inclusive).
  - Write a for loop that computes the total of the integers from 1 to n.
  - How would you modify the loop of the `investment.py` program to print all balances until the investment has doubled?

**Practice It** Now you can try these exercises at the end of the chapter: R4.18, R4.19, P4.8.

**Optional science, graphics, and business exercises** engage students with realistic applications.

**Business P4.28** *Currency conversion.* Write a program that first asks the user to type today's price for one dollar in Japanese yen, then reads U.S. dollar values and converts each to yen. Use 0 as a sentinel.

CANADA	CAD	0.9512	0.8887
CHINA	CNY	7.7769	6.8910
EURO	EUR	0.6644	0.6100
JAPAN	JPY	110.800	100.000

**Graphics P2.30** Write a program that displays the Olympic rings. Color the rings in the Olympic colors.

**Science P4.37** Radioactive decay of radioactive materials can be modeled by the equation  $A = A_0 e^{-t(\log 2/b)}$  where  $A$  is the amount of the material at time  $t$ ,  $A_0$  is the amount at time 0, and  $b$  is the half-life.

Technetium-99 is a radioisotope that is used in imaging of the brain. It has a half-life of 6 hours. Your program should display the relative amount  $A/A_0$  in a patient body every hour for 24 hours after receiving a dose.

```

ch04/doubleInv.py
1 ##
2 # This program computes the time required to double an investment.
3 #
4
5 # Create constant variables.
6 RATE = 5.0
7 INITIAL_BALANCE = 10000.0
8 TARGET = 2 * INITIAL_BALANCE
9
10 # Initialize variables used with the loop.
11 balance = INITIAL_BALANCE
12 year = 0
13
14 # Count the years required for the investment to double.
15 while balance < TARGET :
16     year = year + 1
17     interest = balance * RATE / 100
18     balance = balance + interest
19
20 # Print the results.
21 print("The investment doubled after", year, "years.")
    
```

**Program listings** are carefully designed for easy reading, going well beyond simple color coding. Methods and functions are set off by a subtle outline.

**Common Errors** describe the kinds of errors that students often make, with an explanation of why the errors occur, and what to do about them.

Common Error 3.2



**Exact Comparison of Floating-Point Numbers**

Floating-point numbers have only a limited precision, and calculations can introduce roundoff errors. You must take these inevitable roundoffs into account when comparing floating-point numbers. For example, the following code multiplies the square root of 2 by itself. Ideally, we expect to get the answer 2:

```
from math import sqrt

r = sqrt(2.0)
if r * r == 2.0 :
    print("sqrt(2.0) squared is 2.0")
else :
    print("sqrt(2.0) squared is not 2.0 but", r * r)
```

This program displays  
sqrt(2.0) squared is not 2.0 but 2.0000000000000004

It does not make sense in most circumstances to compare floating-point numbers exactly. Instead, we should test whether they are *close enough*. That is, the magnitude of their difference should be less than some threshold. Mathematically, we would write that  $x$  and  $y$  are close



Take limited precision into account when comparing floating-point numbers.

Programming Tip 3.2



**Hand-Tracing**

A very useful technique for understanding whether a program works correctly is called *hand-tracing*. You simulate the program's activity on a sheet of paper. You can use this method with pseudocode or Python code.

Get an index card, a cocktail napkin, or whatever sheet of paper is within reach. Make a column for each variable. Have the program code ready. Use a marker, such as a paper clip, to mark the current statement. In your mind, execute statements one at a time. Every time the value of a variable changes, cross out the old value and write the new value below the old one.

Let's trace the `taxes.py` program on page 107 with the inputs from the program run that follows it. In lines 12 and 13, `income` and `maritalStatus` are initialized by input statements.

```
5 # Initialize constant variables for the tax rates and rate limits.
6 RATE1 = 0.10
7 RATE2 = 0.25
8 RATE1_SINGLE_LIMIT = 32000.0
9 RATE1_MARRIED_LIMIT = 64000.0
10
11 # Read income and marital status.
12 income = float(input("Please enter your income: "))
13 maritalStatus = input("Please enter s for single, m for married: ")
```

In lines 16 and 17, `tax1` and `tax2` are initialized to 0.0.

```
16 tax1 = 0.0
17 tax2 = 0.0
```



Hand-tracing helps you understand whether a program works correctly.

tax1	tax2	income	marital status
		80000	m

tax1	tax2	income	marital status
0	0	80000	m

**Programming Tips** explain good programming practices, and encourage students to be more productive with tips and techniques such as hand-tracing.

**Computing & Society** presents social and historical information on computing—for interest and to fulfill the “historical and social context” requirements of the ACM/IEEE curriculum guidelines.



**Computing & Society 1.1** Computers Are Everywhere

When computers were first invented in the 1940s, a computer filled an entire room. The photo below shows the ENIAC (electronic numerical integrator and computer), completed in 1946 at the University of Pennsylvania. The ENIAC was used by the military to compute the trajectories of projectiles. Nowadays, computing facilities of search engines, internet shops, and social networks fill huge buildings called data centers. At the other end of the spectrum, computers are all around us. Your cell phone has a computer inside, as do many credit cards and fare cards for public transit. A modern car has several computers—to control the engine, brakes, lights, and the radio.

The advent of ubiquitous computing changed many aspects of our lives. Factories used to employ people to do repetitive assembly tasks that are today carried out by computer-controlled robots, operated by a few people who know how to work with those computers. Books, music, and movies nowadays are often consumed on computers, and computers are almost always involved in their production. The book that you are reading right now could not have been written without computers.



This transit card contains a computer.

**Special Topics** present optional topics and provide additional explanation of others.

Special Topic 3.4



**Short-Circuit Evaluation of Boolean Operators**

The `and` or `or` operators are computed using **short-circuit evaluation**. In other words, logical expressions are evaluated from left to right, and evaluation stops as soon as the truth value is determined. When an `and` is evaluated and the first condition is false, the second condition is not evaluated, because it does not matter what the outcome of the second test is.

For example, consider the expression  
`quantity > 0 and price / quantity < 10`  
Suppose the value of `quantity` is zero. Then the test `quantity > 0` fails, and the second test is not attempted. That is just as well, because it is illegal to divide by zero.

Similarly, when the first condition of an `or` expression is true, then the remainder is not evaluated because the result must be true.

The `and` and `or` operators are computed using **short-circuit evaluation**. As soon as the truth value is determined, no further conditions are evaluated.

In a short circuit, electricity travels along the path of least resistance. Similarly, short-circuit evaluation takes the fastest path for computing the result of a Boolean expression.



## Acknowledgments

Many thanks to Beth Lang Golub, Don Fowley, Katherine Willis, Katie Singleton, Lisa Gee, and Sujin Hong at John Wiley & Sons, and Vickie Piercey at Publishing Services for their help with this project. An especially deep acknowledgment and thanks goes to Cindy Johnson for her hard work, sound judgment, and amazing attention to detail.

We are grateful to Ben Stephenson, University of Calgary, for his excellent work on the supplemental material.

Many thanks to the individuals who provided feedback, reviewed the manuscript, made valuable suggestions, and brought errors and omissions to our attention. They include:

Claude Anderson, *Rose Hulman Institute of Technology*

Gokcen Cilingir, *Washington State University*

Dirk Grunwald, *University of Colorado Boulder*

Andrew Harrington, *Loyola University Chicago*

Debbie Keen, *University of Kentucky*

Nicholas A. Kraft, *University of Alabama*

Aaron Langille, *Laurentian University*

Shyamal Mitra, *University of Texas Austin*

John Schneider, *Washington State University*

Amit Singhal, *University of Rochester*

Ben Stephenson, *University of Calgary*

Dave Sullivan, *Boston University*

Jay Summet, *Georgia Institute of Technology*

James Tam, *University of Calgary*

Krishnaprasad Thirunarayan, *Wright State University*

Peter Tucker, *Whitworth University*

Frances VanScoy, *West Virginia University*

# CONTENTS

PREFACE	v
SPECIAL FEATURES	xviii

## CHAPTER 1 INTRODUCTION 1

1.1	Computer Programs	2
1.2	The Anatomy of a Computer	3
1.3	The Python Programming Language	5
1.4	Becoming Familiar with Your Programming Environment	6
1.5	Analyzing Your First Program	11
1.6	Errors	14
1.7	Problem Solving: Algorithm Design	16

## CHAPTER 2 PROGRAMMING WITH NUMBERS AND STRINGS 29

2.1	Variables	30
2.2	Arithmetic	37
2.3	Problem Solving: First Do It By Hand	45
2.4	Strings	48
2.5	Input and Output	55
2.6	Graphics: Simple Drawings	65

## CHAPTER 3 DECISIONS 91

3.1	The if Statement	92
3.2	Relational Operators	97
3.3	Nested Branches	106
3.4	Multiple Alternatives	109
3.5	Problem Solving: Flowcharts	112
3.6	Problem Solving: Test Cases	116
3.7	Boolean Variables and Operators	118
3.8	Analyzing Strings	124
3.9	Application: Input Validation	127

**CHAPTER 4**    **LOOPS**    **155**

- 4.1 The while Loop    **156**
- 4.2 Problem Solving: Hand-Tracing    **163**
- 4.3 Application: Processing Sentinel Values    **166**
- 4.4 Problem Solving: Storyboards    **170**
- 4.5 Common Loop Algorithms    **173**
- 4.6 The for Loop    **177**
- 4.7 Nested Loops    **184**
- 4.8 Processing Strings    **190**
- 4.9 Application: Random Numbers and Simulations    **194**

**CHAPTER 5**    **FUNCTIONS**    **219**

- 5.1 Functions as Black Boxes    **220**
- 5.2 Implementing and Testing Functions    **222**
- 5.3 Parameter Passing    **226**
- 5.4 Return Values    **229**
- 5.5 Functions Without Return Values    **237**
- 5.6 Problem Solving: Reusable Functions    **239**
- 5.7 Problem Solving: Stepwise Refinement    **242**
- 5.8 Variable Scope    **251**
- 5.9 Recursive Functions (Optional)    **258**

**CHAPTER 6**    **LISTS**    **277**

- 6.1 Basic Properties of Lists    **278**
- 6.2 List Operations    **284**
- 6.3 Common List Algorithms    **290**
- 6.4 Using Lists with Functions    **297**
- 6.5 Problem Solving: Adapting Algorithms    **303**
- 6.6 Problem Solving: Discovering Algorithms by Manipulating Physical Objects    **310**
- 6.7 Tables    **314**



**CHAPTER 7** FILES AND EXCEPTIONS **341**

- 7.1 Reading and Writing Text Files **342**
- 7.2 Text Input and Output **346**
- 7.3 Command Line Arguments **357**
- 7.4 Binary Files and Random Access (Optional) **368**
- 7.5 Exception Handling **377**
- 7.6 Application: Handling Input Errors **383**

**CHAPTER 8** SETS AND DICTIONARIES **403**

- 8.1 Sets **404**
- 8.2 Dictionaries **414**
- 8.3 Complex Structures **424**

**CHAPTER 9** OBJECTS AND CLASSES **443**

- 9.1 Object-Oriented Programming **444**
- 9.2 Implementing a Simple Class **446**
- 9.3 Specifying the Public Interface of a Class **450**
- 9.4 Designing the Data Representation **452**
- 9.5 Constructors **454**
- 9.6 Implementing Methods **457**
- 9.7 Testing a Class **461**
- 9.8 Problem Solving: Tracing Objects **469**
- 9.9 Problem Solving: Patterns for Object Data **472**
- 9.10 Object References **478**
- 9.11 Application: Writing a Fraction Class **482**

**CHAPTER 10** INHERITANCE **507**

- 10.1 Inheritance Hierarchies **508**
- 10.2 Implementing Subclasses **513**
- 10.3 Calling the Superclass Constructor **517**
- 10.4 Overriding Methods **521**
- 10.5 Polymorphism **524**
- 10.6 Application: A Geometric Shape Class Hierarchy **538**

**CHAPTER 11 RECURSION 555**

- 11.1 Triangle Numbers Revisited **556**
- 11.2 Problem Solving: Thinking Recursively **560**
- 11.3 Recursive Helper Functions **565**
- 11.4 The Efficiency of Recursion **566**
- 11.5 Permutations **571**
- 11.6 Backtracking **575**
- 11.7 Mutual Recursion **583**

**CHAPTER 12 SORTING AND SEARCHING 597**

- 12.1 Selection Sort **598**
- 12.2 Profiling the Selection Sort Algorithm **600**
- 12.3 Analyzing the Performance of the Selection Sort Algorithm **602**
- 12.4 Merge Sort **606**
- 12.5 Analyzing the Merge Sort Algorithm **609**
- 12.6 Searching **614**
- 12.7 Problem Solving: Estimating the Running Time of an Algorithm **617**

**APPENDICES**

- APPENDIX A** THE BASIC LATIN AND LATIN-1 SUBSETS OF UNICODE **A-1**
- APPENDIX B** PYTHON OPERATOR SUMMARY **A-4**
- APPENDIX C** PYTHON RESERVED WORD SUMMARY **A-6**
- APPENDIX D** THE PYTHON STANDARD LIBRARY **A-8**
- APPENDIX E** BINARY NUMBERS AND BIT OPERATIONS **A-29**

GLOSSARY **G-1**

INDEX **I-1**

CREDITS **C-1**

**ALPHABETICAL LIST OF SYNTAX BOXES**

Assignment	31
Calling Functions	40
Constructor	455
for Statement	178
for Statement with range Function	179
Function Definition	223
Handling Exceptions	379
if Statement	94
Lists	279
Method Definition	458
Opening and Closing Files	343
print Statement	12
Program with Functions	224
Raising an Exception	378
Set and Dictionary Literals	415
String Format Operator	57
Subclass Constructor	517
Subclass Definition	514
The finally Clause	381
while Statement	157

**CHAPTER**

**Common Errors**

**How Tos and Worked Examples**

<b>1</b> Introduction	Misspelling Words	15	Describing an Algorithm with Pseudocode	19
			Writing an Algorithm for Tiling a Floor	20
<b>2</b> Programming with Numbers and Strings	Using Undefined Variables	36	Computing Travel Time	47
	Roundoff Errors	43	Writing Simple Programs	60
	Unbalanced Parentheses	43	Computing the Cost of Stamps	63
			Drawing Graphical Shapes	72
<b>3</b> Decisions	Tabs	96	Implementing an if Statement	102
	Exact Comparison of Floating-Point Numbers	101	Extracting the Middle	104
	Confusing and or Conditions	121	Intersecting Circles	131
<b>4</b> Loops	Don't Think "Are We There Yet?"	160	Writing a Loop	182
	Infinite Loops	161	Average Exam Grades	188
	Off-by-One Errors	161	Bull's Eye	197
<b>5</b> Functions	Trying to Modify Arguments	228	Implementing a Function	231
			Generating Random Passwords	233
			Calculating a Course Grade	248
			Rolling Dice	254
			Thinking Recursively	260



## Programming Tips



## Special Topics



## Random Facts

Interactive Mode	9	The Python Interpreter	10	Computers Are Everywhere	5
Backup Copies	10				
Choose Descriptive Variable Names	36	Other Ways to Import Modules	44	International Alphabets and Unicode	54
Do Not Use Magic Numbers	37	Combining Assignment and Arithmetic	44	The Pentium Floating-Point Bug	65
Use Spaces in Expressions	44	Line Joining	45		
Don't Wait to Convert	60	Character Values	53		
		Escape Sequences	54		
Avoid Duplication in Branches	96	Conditional Expressions	97	Denver's Luggage Handling System	116
Hand-Tracing	108	Lexicographic Ordering of Strings	101	Artificial Intelligence	135
Make a Schedule and Make Time for Unexpected Problems	117	Chaining Relational Operators	122		
Readability	122	Short-Circuit Evaluation of Boolean Operators	123		
		De Morgan's Law	123		
		Terminating a Program	130		
		Text Input in Graphical Programs	131		
Count Iterations	181	Processing Sentinel Values with a Boolean Variable	169	The First Bug	162
		Redirection of Input and Output	169	Software Piracy	200
		Special Form of the print Function	188		
Function Comments	226	Using Single-Line Compound Statements	230	Personal Computing	257
Do Not Modify Parameter Variables	228				
Keep Functions Short	246				
Tracing Functions	247				
Stubs	248				
Avoid Global Variables	253				

## CHAPTER

Common  
ErrorsHow Tos  
and  
Worked Examples**6** Lists

Out-of-Range Errors 282

Working with Lists	304
Rolling the Dice	306
A World Population Table	319
Drawing Regular Polygons	322

**7** Files and Exceptions

Backslashes in File Names 346

Processing Text Files	360
Analyzing Baby Names	363
Displaying a Scene File	387

**8** Sets and Dictionaries

Unique Words	411
Translating Text Messages	422
Pie Charts	430

**9** Objects and Classes

Trying to Call a Constructor 456

Implementing a Class	463
Implementing a Bank Account Class	466
A Die Class	491

**10** Inheritance

Confusing Super- and Subclasses	516
Forgetting to Use the super Function When Invoking a Superclass Method	524
Don't Use Type Tests	530

Developing an Inheritance Hierarchy	530
Implementing an Employee Hierarchy for Payroll Processing	535

**11** Recursion

Infinite Recursion 559

Finding Files	564
Towers of Hanoi	580

**12** Sorting and Searching

Enhancing the Insertion Sort Algorithm	623
---	-----



## Programming Tips



## Special Topics



## Random Facts

Use Lists for Sequences of Related Items	283	Reverse Subscripts	282	Computer Viruses	283
		Slices	290		
		Call by Value and Call by Reference	300		
		Tuples	301		
		Functions with a Variable Number of Arguments	301		
		Tuple Assignment	302		
		Returning Multiple Values with Tuples	302		
		Tables with Variable Row Lengths	321		
Raise Early, Handle Late	382	Reading the Entire File	355	Encryption Algorithms	366
Do Not Use except and finally in the Same try Statement	382	Regular Expressions	355	The Ariane Rocket Incident	390
		Character Encodings	356		
		Reading Web Pages	357		
		The with Statement	383		
Use Python Sets, Not Lists, for Efficient Set Operations	412	Hashing	413	Standardization	414
		Iterating over Dictionary Items	421		
		Storing Data Records	421		
		User Modules	430		
Make all Instance Variables Private, Most Methods Public	453	Default and Named Arguments	456	Electronic Voting Machines	477
Define Instance Variables Only in the Constructor	460	Class Variables	460	Open Source and Free Software	494
		Object Types and Instances	490		
Use a Single Class for Variation in Values, Inheritance for Variation in Behavior	511	The Cosmic Superclass: object	512		
		Subclasses and Instances	528		
		Dynamic Method Lookup	528		
		Abstract Classes	529		
		Recursion with Objects	560	The Limits of Computation	574
Searching and Sorting	622	Oh, Omega, and Theta	604	The First Programmer	613
		Insertion Sort	605		
		The Quicksort Algorithm	611		
		Comparing Objects	623		





# CHAPTER 1

## INTRODUCTION



### CHAPTER GOALS

- To learn about computers and programming
- To write and run your first Python program
- To recognize compile-time and run-time errors
- To describe an algorithm with pseudocode

### CHAPTER CONTENTS

- 1.1 COMPUTER PROGRAMS** 2
- 1.2 THE ANATOMY OF A COMPUTER** 3
  - Computing & Society 1.1: Computers Are Everywhere* 5
- 1.3 THE PYTHON PROGRAMMING LANGUAGE** 5
- 1.4 BECOMING FAMILIAR WITH YOUR PROGRAMMING ENVIRONMENT** 6
  - Programming Tip 1.1: Interactive Mode* 9
  - Programming Tip 1.2: Backup Copies* 10
  - Special Topic 1.1: The Python Interpreter* 10

- 1.5 ANALYZING YOUR FIRST PROGRAM** 11
  - Syntax 1.1: print Statement* 12
- 1.6 ERRORS** 14
  - Common Error 1.1: Misspelling Words* 15
- 1.7 PROBLEM SOLVING: ALGORITHM DESIGN** 16
  - How To 1.1: Describing an Algorithm with Pseudocode* 19
  - Worked Example 1.1: Writing an Algorithm for Tiling a Floor* 20



Just as you gather tools, study a project, and make a plan for tackling it, in this chapter you will gather up the basics you need to start learning to program. After a brief introduction to computer hardware, software, and programming in general, you will learn how to write and run your first Python program. You will also learn how to diagnose and fix programming errors, and how to use pseudocode to describe an *algorithm*—a step-by-step description of how to solve a problem—as you plan your computer programs.

## 1.1 Computer Programs

Computers execute very basic instructions in rapid succession.

You have probably used a computer for work or fun. Many people use computers for everyday tasks such as electronic banking or writing a term paper. Computers are good for such tasks. They can handle repetitive chores, such as totaling up numbers or placing words on a page, without getting bored or exhausted.

The flexibility of a computer is quite an amazing phenomenon. The same machine can balance your checkbook, lay out your term paper, and play a game. In contrast, other machines carry out a much narrower range of tasks; a car drives and a toaster toasts. Computers can carry out a wide range of tasks because they execute different programs, each of which directs the computer to work on a specific task.

A computer program is a sequence of instructions and decisions.

The computer itself is a machine that stores data (numbers, words, pictures), interacts with devices (the monitor, the sound system, the printer), and executes programs. A **computer program** tells a computer, in minute detail, the sequence of steps that are needed to fulfill a task. The physical computer and peripheral devices are collectively called the **hardware**. The programs the computer executes are called the **software**.

Today's computer programs are so sophisticated that it is hard to believe that they are composed of extremely primitive instructions. A typical instruction may be one of the following:

- Put a red dot at a given screen position.
- Add up two numbers.
- If this value is negative, continue the program at a certain instruction.

The computer user has the illusion of smooth interaction because a program contains a huge number of such instructions, and because the computer can execute them at great speed.

The act of designing and implementing computer programs is called **programming**. In this book, you will learn how to program a computer—that is, how to direct the computer to execute tasks.

Programming is the act of designing and implementing computer programs.

To write a computer game with motion and sound effects or a word processor that supports fancy fonts and pictures is a complex task that requires a team of many highly-skilled programmers. Your first programming efforts will be more mundane. The concepts and skills you learn in this book form an important foundation, and you should not be disappointed if your first programs do not rival the sophisticated software that is familiar to you. Actually, you will find that there is an immense thrill even in simple programming tasks. It is an amazing experience to see the computer precisely and quickly carry out a task that would take you hours of drudgery, to

make small changes in a program that lead to immediate improvements, and to see the computer become an extension of your mental powers.



1. What is required to play music on a computer?
2. Why is a CD player less flexible than a computer?
3. What does a computer user need to know about programming in order to play a video game?

## 1.2 The Anatomy of a Computer

To understand the programming process, you need to have a rudimentary understanding of the building blocks that make up a computer. We will look at a personal computer. Larger computers have faster, larger, or more powerful components, but they have fundamentally the same design.

At the heart of the computer lies the **central processing unit (CPU)** (see Figure 1). The inside wiring of the CPU is enormously complicated. The CPUs used for personal computers at the time of this writing are composed of several hundred million structural elements, called *transistors*.

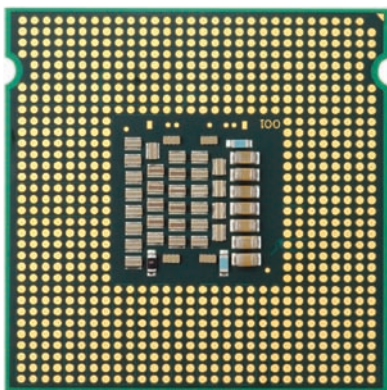
The central processing unit (CPU) performs program control and data processing.

The CPU performs program control and data processing. That is, the CPU locates and executes the program instructions; it carries out arithmetic operations such as addition, subtraction, multiplication, and division; it fetches data from external memory or devices and places processed data into storage.

Storage devices include memory and secondary storage.

There are two kinds of storage. **Primary storage** is made from memory chips: electronic circuits that can store data, provided they are supplied with electric power. **Secondary storage**, usually a **hard disk** (see Figure 2), provides slower and less expensive storage that persists without electricity. A hard disk consists of rotating platters, which are coated with a magnetic material, and read/write heads, which can detect and change the magnetic flux on the platters.

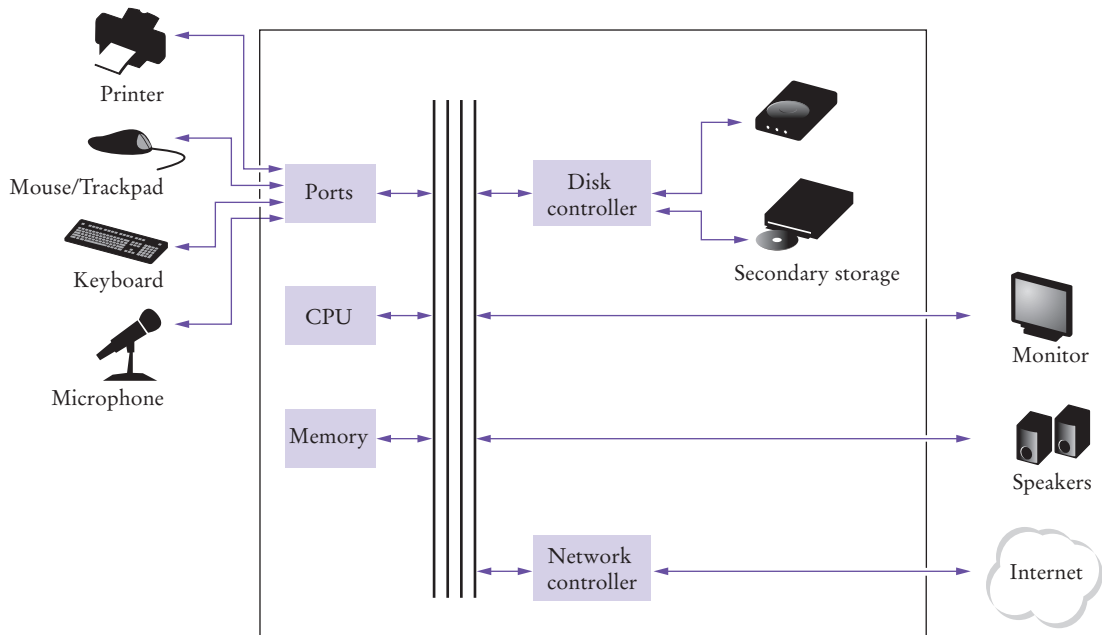
The computer stores both data and programs. They are located in secondary storage and loaded into memory when the program starts. The program then updates the data in memory and writes the modified data back to secondary storage.



**Figure 1** Central Processing Unit



**Figure 2** A Hard Disk



**Figure 3** Schematic Design of a Personal Computer

To interact with a human user, a computer requires peripheral devices. The computer transmits information (called *output*) to the user through a display screen, speakers, and printers. The user can enter information (called *input*) for the computer by using a keyboard or a pointing device such as a mouse.

Some computers are self-contained units, whereas others are interconnected through **networks**. Through the network cabling, the computer can read data and programs from central storage locations or send data to other computers. To the user of a networked computer, it may not even be obvious which data reside on the computer itself and which are transmitted through the network.

Figure 3 gives a schematic overview of the architecture of a personal computer. Program instructions and data (such as text, numbers, audio, or video) are stored on the hard disk, on a compact disk (or DVD), or elsewhere on the network. When a program is started, it is brought into memory, where the CPU can read it. The CPU reads the program one instruction at a time. As directed by these instructions, the CPU reads data, modifies it, and writes it back to memory or the hard disk. Some program instructions will cause the CPU to place dots on the display screen or printer or to vibrate the speaker. As these actions happen many times over and at great speed, the human user will perceive images and sound. Some program instructions read user input from the keyboard or mouse. The program analyzes the nature of these inputs and then executes the next appropriate instruction.



4. Where is a program stored when it is not currently running?
5. Which part of the computer carries out arithmetic operations, such as addition and multiplication?

**Practice It** Now you can try these exercises at the end of the chapter: R1.2, R1.3.



## Computing & Society 1.1 Computers Are Everywhere

When computers were first invented in the 1940s, a computer filled an entire room. The photo below shows the ENIAC (electronic numerical integrator and computer), completed in 1946 at the University of Pennsylvania. The ENIAC was used by the military to compute the trajectories of projectiles. Nowadays, computing facilities of search engines, Internet shops, and social networks fill huge buildings called data centers. At the other end of the spectrum, computers are all around us. Your cell phone has a computer inside, as do many credit cards and fare cards for public transit. A modern car has several computers—to control the engine, brakes, lights, and the radio.

The advent of ubiquitous computing changed many aspects of our lives. Factories used to employ people to do repetitive assembly tasks that are today carried out by computer-controlled robots, operated by a few people who know how to work with those computers. Books, music, and movies nowadays are often consumed on computers, and computers are almost always involved in their production. The book that you are reading right now could

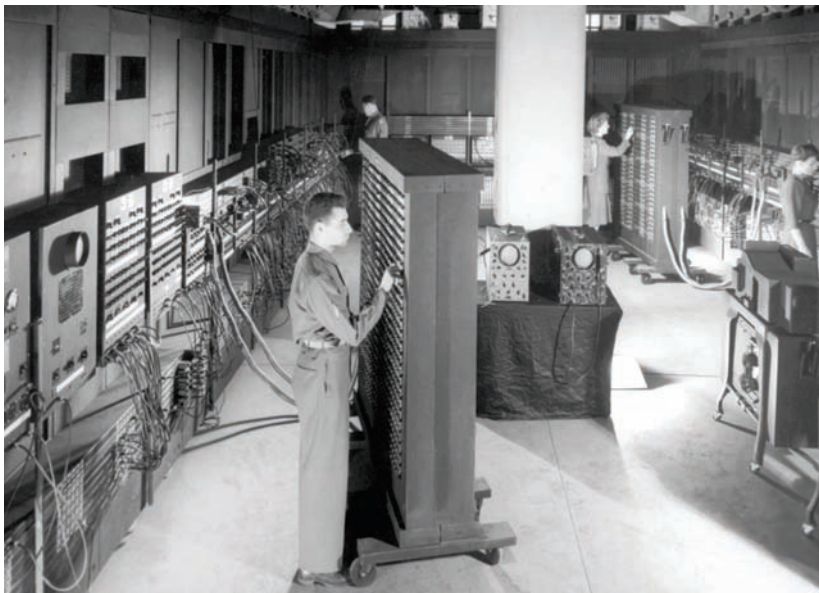


*This transit card contains a computer.*

not have been written without computers.

Knowing about computers and how to program them has become an essential skill in many careers. Engineers design computer-controlled cars and medical equipment that preserve lives. Computer scientists develop programs that help people come together to support social causes. For example, activists used social networks to share videos showing abuse by repressive regimes, and this information was instrumental in changing public opinion.

As computers, large and small, become ever more embedded in our everyday lives, it is increasingly important for everyone to understand how they work, and how to work with them. As you use this book to learn how to program a computer, you will develop a good understanding of computing fundamentals that will make you a more informed citizen and, perhaps, a computing professional.



*The ENIAC*

## 1.3 The Python Programming Language

In order to write a computer program, you need to provide a sequence of instructions that the CPU can execute. A computer program consists of a large number of simple CPU instructions, and it is tedious and error-prone to specify them one by one. For that reason, **high-level programming languages** have been created. These languages

allow a programmer to specify the desired program actions at a high level. The high-level instructions are then automatically translated into the more detailed instructions required by the CPU.

In this book, we will use a high-level programming language called Python, which was developed in the early 1990s by Guido van Rossum. Van Rossum needed to carry out repetitive tasks for administering computer systems. He was dissatisfied with other available languages that were optimized for writing large and fast programs. He needed to write smaller programs that didn't have to run at optimum speed. It was important to him that he could author the programs quickly and update them quickly as his needs changed. Therefore, he designed a language that made it very easy to work with complex data. Python has evolved considerably since its beginnings. In this book, we use version 3 of the Python language. Van Rossum is still the principal author of the language, but the effort now includes many volunteers.



Python is portable and easy to learn and use.

Python has become popular for business, scientific, and academic applications and is very suitable for the beginning programmer. There are many reasons for the success of Python. Python has a much simpler and cleaner syntax than other popular languages such as Java, C, and C++, which makes it easier to learn. Moreover, you can try out short Python programs in an interactive environment, which encourages experimentation and rapid turnaround. Python is also very portable between computer systems. The same Python program will run, without change, on Windows, UNIX, Linux, or Macintosh.



6. Why don't you specify a program directly in CPU instructions?
7. What are the two most important benefits of the Python language?

**Practice It** Now you can try this exercise at the end of the chapter: R1.5.

## 1.4 Becoming Familiar with Your Programming Environment

Set aside some time to become familiar with the programming environment that you will use for your class work.

Many students find that the tools they need as programmers are very different from the software with which they are familiar. You should spend some time making yourself familiar with your programming environment. Because computer systems vary widely, this book can only give an outline of the steps you need to follow. It is a good idea to participate in a hands-on lab, or to ask a knowledgeable friend to give you a tour.

A text editor is a program for entering and modifying text, such as a Python program.

### Step 1 Start the Python development environment.

Computer systems differ greatly in this regard. On many computers there is an **integrated development environment** in which you can write and test your programs. On other computers you first launch a **text editor**, a program that functions like a word processor, in which you can enter your Python instructions; you then open a **terminal window** and type commands to execute your program. You need to find out how to get started with the development environment you will use to write code in Python.

### Step 2 Write a simple program.

The traditional choice for the very first program in a new programming language is a program that displays a simple greeting: “Hello, World!”. Let us follow that tradition. Here is the “Hello, World!” program in Python:

```
# My first Python program.
print("Hello, World!")
```

We will examine this program in the next section.

No matter which programming environment you use, you begin your activity by typing the program instructions into an editor window.

Create a new file and call it `hello.py`, using the steps that are appropriate for your environment. (If your environment requires that you supply a project name in addition to the file name, use the name `hello` for the project.) Enter the program instructions *exactly* as they are given above. Alternatively, locate the electronic copy in this book’s companion code and paste it into your editor.

Python is case sensitive. You must be careful about distinguishing between upper- and lowercase letters.

As you write this program, pay careful attention to the various symbols, and keep in mind that Python is **case sensitive**. You must enter upper- and lowercase letters exactly as they appear in the program listing. You cannot type `Print` or `PRINT`. If you are not careful, you will run into problems—see Common Error 1.1 on page 15.

### Step 3 Run the program.

The process for running a program depends greatly on your programming environment. You may have to click a button or enter some commands. When you run the test program, the message

```
Hello, World!
```

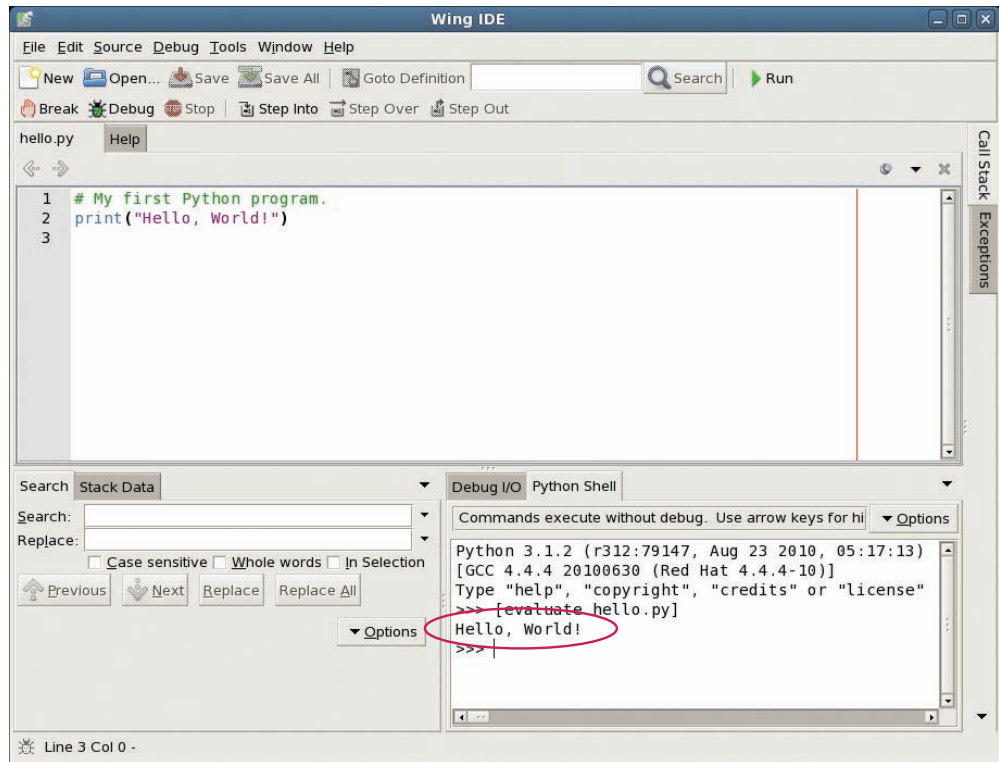
will appear somewhere on the screen (see Figures 4 and 5).



```
Terminal
~/PythonForEveryone$ cd ch01
~/PythonForEveryone/ch01$ python hello.py
Hello, World!
~/PythonForEveryone/ch01$
```

**Figure 4** Running the `hello.py` Program in a Terminal Window

**Figure 5**  
Running the  
hello.py Program  
in an Integrated  
Development  
Environment



The Python interpreter reads Python programs and executes the program instructions.

A Python program is executed using the **Python interpreter**. The interpreter reads your program and executes all of its steps. (Special Topic 1.1 on page 10 explains in more detail what the Python interpreter does.) In some programming environments, the Python interpreter is automatically launched when you click on a “Run” button or select the “Run” option from a menu. In other environments, you have to launch the interpreter explicitly.

#### Step 4 Organize your work.

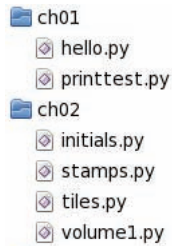
As a programmer, you write programs, try them out, and improve them. If you want to keep your programs, or turn them in for grading, you store them in **files**. A Python program can be stored in a file with any name, provided it ends with `.py`. For example, we can store our first program in a file named `hello.py` or `welcome.py`.

Files are stored in **folders** or **directories**. A folder can contain files as well as other folders, which themselves can contain more files and folders (see Figure 6). This hierarchy can be quite large, and you need not be concerned with all of its branches. However, you should create folders for organizing your work. It is a good idea to make a separate folder for your programming class. Inside that folder, make a separate folder for each program.

Some programming environments place your programs into a default location if you don’t specify a folder. In that case, you need to find out where those files are located.

Be sure that you understand where your files are located in the folder hierarchy. This information is essential when you submit files for grading, and for making backup copies (see Programming Tip 1.2 on page 10).



**Figure 6**  
A Folder Hierarchy**SELF CHECK**

8. Where is the `hello.py` file stored on your computer?
9. What do you do to protect yourself from data loss when you work on programming projects?

**Practice It** Now you can try this exercise at the end of the chapter: R1.6.

## Programming Tip 1.1

**Interactive Mode**

When you write a complete program, you place the program instructions in a file and let the Python interpreter execute your program file. The interpreter, however, also provides an interactive mode in which Python instructions can be entered one at a time. To launch the Python interactive mode from a terminal window, enter the command

```
python
```

(On systems where multiple versions of Python are installed, use the command `python3` to run version 3 of Python.) Interactive mode can also be started from within most Python integrated development environments.

The interface for working in interactive mode is known as the **Python shell**. First, you will see an informational message similar to the following:

```
Python 3.1.2 (r312:79147, Aug 23 2010, 05:17:13)
[GCC 4.4.4 20100630 (Red Hat 4.4.4-10)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The `>>>` at the bottom of the output is the **prompt**. It indicates that you can enter Python instructions. After you type an instruction and press the Enter key, the code is immediately executed by the Python interpreter. For example, if you enter

```
print("Hello, World!")
```

the interpreter will respond by executing the `print` function and displaying the output, followed by another prompt:

```
>>> print("Hello, World!")
Hello World
>>>
```

Interactive mode is very useful when you are first learning to program. It allows you to experiment and test individual Python instructions to see what happens. You can also use interactive mode as a simple calculator. Just enter mathematical expressions using Python syntax:

```
>>> 7035 * 0.15
1055.25
>>>
```

Make it a habit to use interactive mode as you experiment with new language constructs.