

Starting Out With Python 6th Edition PDF

Visit the link below to download the full version of the ebook

DOWNLOAD NOW

starting out with >>>

PYTHON[®]

SIXTH EDITION



TONY G.



Scan to Download
or Type the Link

ebook.ac/starting6e

starting out with >>>

PYTHON®

SIXTH EDITION



TONY GADDIS

Contents

Welcome

Cover	11
Title Page	11
Copyright Page	11
Pearson’s Commitment to Diversity, Equity, and Inclusion	13
Contents in a Glance	
Contents in a Glance.....	13
Location of VideoNotes.....	14
Preface	
Preface	16
Changes in the Sixth Edition	16
Brief Overview of Each Chapter	17
Organization of the Text	20
Features of the Text.....	21
Supplements.....	22
Acknowledgments.....	23
About the Author.....	25
Ordering Options.....	25

1: Introduction to Computers and Programming

1: Topics.....	27
1.1: Introduction	
1.1: Introduction.....	27
1.2: Hardware and Software	
1.2: Hardware and Software.....	28
Main Memory.....	31
Software	32
1.2: (Noninteractive) Checkpoint Questions from the Book	33
1.3: How Computers Store Data	
1.3: How Computers Store Data.....	33
1.3: (Noninteractive) Checkpoint Questions from the Book	38
1.4: How a Program Works	
1.4: How a Program Works.....	38
From Machine Language to Assembly Language	40

Keywords, Operators, and Syntax: An Overview.....	43
1.4: (Noninteractive) Checkpoint Questions from the Book	45

1.5: Using Python

1.5: Using Python	46
Writing Python Programs and Running Them in Script Mode	48
The IDLE Programming Environment	49

Chapter 1: From the Book for Additional Practice

Chapter 1: Review Questions	50
Chapter 1: Programming Exercises	54

2: Input, Processing, and Output

2: Topics.....	56
2.1: Designing a Program	
2.1: Designing a Program.....	56
More About the Design Process.....	57
2.1: (Noninteractive) Checkpoint Questions from the Book	60
2.2: Input, Processing, and Output	
2.2: Input, Processing, and Output	61
2.3: Displaying Output with the print Function	
2.3: Displaying Output with the print Function ...	61
2.3: (Noninteractive) Checkpoint Questions from the Book	64
2.4: Comments	
2.4: Comments.....	64
2.5: Variables	
2.5: Variables	65
Creating Variables with Assignment Statements	66
Multiple Assignment.....	68
Variable Naming Rules	69
Displaying Multiple Items with the print Function	70
Variable Reassignment	71
Numeric Data Types and Literals	72
Storing Strings with the str Data Type	73

Reassigning a Variable to a Different Type	74	2.11: Named Constants.....	108
2.5: (Noninteractive) Checkpoint Questions from the Book	75	2.11: (Noninteractive) Checkpoint Questions from the Book	109
2.6: Reading Input from the Keyboard		2.12: Introduction to Turtle Graphics	
2.6: Reading Input from the Keyboard.....	75	2.12: Introduction to Turtle Graphics	109
Reading Numbers with the input Function.....	77	Setting the Turtle's Heading to a Specific Angle .	116
2.6: (Noninteractive) Checkpoint Questions from the Book	80	Changing the Drawing Color	120
2.7: Performing Calculations		Moving the Turtle to a Specific Location	121
2.7: Performing Calculations	80	Controlling the Turtle's Animation Speed	123
In the Spotlight: Calculating a Percentage.....	81	Filling Shapes	125
Floating-Point and Integer Division.....	83	Getting Input with a Dialog Box.....	128
Operator Precedence	83	In the Spotlight: The Orion Constellation Program.....	131
In the Spotlight: Calculating an Average.....	85	2.12: (Noninteractive) Checkpoint Questions from the Book	139
The Exponent Operator.....	86	Chapter 2: From the Book for Additional Practice	
The Remainder Operator	87	Chapter 2: Review Questions	139
Converting Math Formulas to Programming Statements.....	88	Chapter 2: Programming Exercises	144
In the Spotlight: Converting a Math Formula to a Programming Statement.....	89	3: Decision Structures and Boolean Logic	
Mixed-Type Expressions and Data Type Conversion.....	90	3: Topics.....	149
Breaking Long Statements into Multiple Lines.....	91	3.1: The if Statement	
2.7: (Noninteractive) Checkpoint Questions from the Book	92	3.1: The if Statement	149
2.8: String Concatenation		Boolean Expressions and Relational Operators .	152
2.8: String Concatenation	92	Putting It All Together	155
2.8: (Noninteractive) Checkpoint Questions from the Book	94	In the Spotlight: Using the if Statement	156
2.9: More About the print Function		Single-Line if Statements	157
2.9: More About the print Function.....	94	3.1: (Noninteractive) Checkpoint Questions from the Book	158
Escape Characters	96	3.2: The if-else Statement	
2.9: (Noninteractive) Checkpoint Questions from the Book	97	3.2: The if-else Statement	158
2.10: Displaying Formatted Output with F-strings		In the Spotlight: Using the if-else Statement	160
2.10: Displaying Formatted Output with F-strings	98	3.2: (Noninteractive) Checkpoint Questions from the Book	161
Placeholder Expressions.....	99	3.3: Comparing Strings	
Formatting Values.....	99	3.3: Comparing Strings	161
Specifying a Minimum Field Width	102	3.3: (Noninteractive) Checkpoint Questions from the Book	165
Aligning Values.....	104	3.4: Nested Decision Structures and the if-elif-else Statement	
The Order of Designators	106	3.4: Nested Decision Structures and the if-elif-else Statement	165
Concatenation with F-strings	106	In the Spotlight: Multiple Nested Decision Structures	170
2.10: (Noninteractive) Checkpoint Questions from the Book	107		

The if-elif-else Statement	172	Single-Line while Loops	215
3.4: (Noninteractive) Checkpoint Questions from the Book	173	4.2: (Noninteractive) Checkpoint Questions from the Book	216
3.5: Logical Operators		4.3: The for Loop: A Count-Controlled Loop	
3.5: Logical Operators	173	4.3: The for Loop: A Count-Controlled Loop	216
The Loan Qualifier Program Revisited	177	Using the range Function with the for Loop	218
Checking Numeric Ranges with Logical Operators	179	Using the Target Variable Inside the Loop	220
3.5: (Noninteractive) Checkpoint Questions from the Book	180	In the Spotlight: Designing a Count-Controlled Loop with the for Statement	222
3.6: Boolean Variables		Letting the User Control the Loop Iterations	224
3.6: Boolean Variables	181	Generating an Iterable Sequence that Ranges from Highest to Lowest	225
3.6: (Noninteractive) Checkpoint Questions from the Book	182	4.3: (Noninteractive) Checkpoint Questions from the Book	226
3.7: Conditional Expressions		4.4: Calculating a Running Total	
3.7: Conditional Expressions	182	4.4: Calculating a Running Total.....	226
3.8: Assignment Expressions and the Walrus Operator		The Augmented Assignment Operators	228
3.8: Assignment Expressions and the Walrus Operator.....	184	4.4: (Noninteractive) Checkpoint Questions from the Book	230
3.9: Turtle Graphics: Determining the State of the Turtle		4.5: Sentinels	
3.9: Turtle Graphics: Determining the State of the Turtle	186	4.5: Sentinels	230
In the Spotlight: The Hit the Target Game	189	In the Spotlight: Using a Sentinel	231
3.9: (Noninteractive) Checkpoint Questions from the Book	193	4.5: (Noninteractive) Checkpoint Questions from the Book	232
Chapter 3: From the Book for Additional Practice		4.6: Input Validation Loops	
Chapter 3: Review Questions	193	4.6: Input Validation Loops	232
Chapter 3: Programming Exercises	197	In the Spotlight: Writing an Input Validation Loop	235
4: Repetition Structures		Using the Walrus Operator in an Input Validation Loop	236
4: Topics.....	204	4.6: (Noninteractive) Checkpoint Questions from the Book	237
4.1: Introduction to Repetition Structures		4.7: Nested Loops	
4.1: Introduction to Repetition Structures.....	204	4.7: Nested Loops.....	237
4.1: (Noninteractive) Checkpoint Questions from the Book	205	In the Spotlight: Using Nested Loops to Print Patterns	240
4.2: The while Loop: A Condition-Controlled Loop		4.8: Using break, continue, and else with Loops	
4.2: The while Loop: A Condition-Controlled Loop	206	4.8: Using break, continue, and else with Loops.....	244
The while Loop Is a Pretest Loop	209	Using the else Clause with a Loop	247
In the Spotlight: Designing a Program with a while Loop	210	4.9: Turtle Graphics: Using Loops to Draw Designs	
Infinite Loops	211	4.9: Turtle Graphics: Using Loops to Draw Designs.....	248
Using the while Loop as a Count-Controlled Loop	212	Chapter 4: From the Book for Additional Practice	

Chapter 4: Review Questions	252	5.7: Introduction to Value-Returning Functions: Generating Random Numbers	295
Chapter 4: Programming Exercises	255	Standard Library Functions and the import Statement	296
5: Functions		Generating Random Numbers	297
5: Topics.....	260	Calling Functions from an F-String	300
5.1: Introduction to Functions		Experimenting with Random Numbers in Interactive Mode	300
5.1: Introduction to Functions	260	In the Spotlight: Using Random Numbers	301
5.1: (Noninteractive) Checkpoint Questions from the Book	262	In the Spotlight: Using Random Numbers to Represent Other Values	302
5.2: Defining and Calling a Void Function		The randrange, random, and uniform Functions	303
5.2: Defining and Calling a Void Function	263	Random Number Seeds	304
Defining and Calling a Function.....	263	5.7: (Noninteractive) Checkpoint Questions from the Book	305
Indentation in Python	267		
5.2: (Noninteractive) Checkpoint Questions from the Book	268	5.8: Writing Your Own Value-Returning Functions	
5.3: Designing a Program to Use Functions		5.8: Writing Your Own Value-Returning Functions	306
5.3: Designing a Program to Use Functions.....	269	How to Use Value-Returning Functions	308
In the Spotlight: Defining and Calling Functions.....	271	Using IPO Charts	310
Pausing Execution Until the User Presses Enter .	274	In the Spotlight: Modularizing with Functions ..	311
Using the pass Keyword	274	Returning Strings	315
5.4: Local Variables		Returning Boolean Values	315
5.4: Local Variables	275	Returning Multiple Values	317
5.4: (Noninteractive) Checkpoint Questions from the Book	277	Returning None from a Function.....	318
5.5: Passing Arguments to Functions		5.8: (Noninteractive) Checkpoint Questions from the Book	320
5.5: Passing Arguments to Functions	277		
In the Spotlight: Passing an Argument to a Function	280	5.9: The math Module	
Passing Multiple Arguments.....	281	5.9: The math Module.....	320
Making Changes to Parameters	283	5.9: (Noninteractive) Checkpoint Questions from the Book	323
Keyword Arguments.....	285		
Keyword-Only Parameters	287	5.10: Storing Functions in Modules	
Positional-Only Parameters	288	5.10: Storing Functions in Modules	323
Default Arguments.....	288	Conditionally Executing the main Function in a Module	326
5.5: (Noninteractive) Checkpoint Questions from the Book	291		
5.6: Global Variables and Global Constants		5.11: Turtle Graphics: Modularizing Code with Functions	
5.6: Global Variables and Global Constants.....	292	5.11: Turtle Graphics: Modularizing Code with Functions	328
In the Spotlight: Using Global Constants.....	294	Storing Your Graphics Functions in a Module ...	332
5.6: (Noninteractive) Checkpoint Questions from the Book	295		
5.7: Introduction to Value-Returning Functions: Generating Random Numbers		Chapter 5: From the Book for Additional Practice	
		Chapter 5: Review Questions	335
		Chapter 5: Programming Exercises	340

6: Files and Exceptions

6: Topics.....	347
6.1: Introduction to File Input and Output	
6.1: Introduction to File Input and Output	347
Types of Files	349
Opening a File	351
Writing Data to a File	352
Reading Data from a File	354
Concatenating a Newline to a String	358
Reading a String and Stripping the Newline from It	360
Appending Data to an Existing File	361
Writing and Reading Numeric Data	362
6.1: (Noninteractive) Checkpoint Questions from the Book	365
6.2: Using Loops to Process Files	
6.2: Using Loops to Process Files	365
Reading a File with a Loop and Detecting the End of the File	366
Using Python's for Loop to Read Lines	368
In the Spotlight: Working with Files	369
6.2: (Noninteractive) Checkpoint Questions from the Book	371
6.3: Using the with Statement to Open Files	
6.3: Using the with Statement to Open Files.....	372
Opening Multiple Files with a with Statement ...	374
6.4: Processing Records	
6.4: Processing Records	375
In the Spotlight: Adding and Displaying Records	379
In the Spotlight: Searching for a Record	380
In the Spotlight: Modifying Records	382
In the Spotlight: Deleting Records	384
6.4: (Noninteractive) Checkpoint Questions from the Book	386
6.5: Exceptions	
6.5: Exceptions	386
Handling Multiple Exceptions	391
Displaying an Exception's Default Error Message	393
The else Clause	395
The finally Clause	396

What If an Exception Is Not Handled? 397

6.5: (Noninteractive) Checkpoint Questions from the Book 397

Chapter 6: From the Book for Additional Practice

Chapter 6: Review Questions 397

Chapter 6: Programming Exercises 401

7: Lists and Tuples

7: Topics..... 404

7.1: Sequences

7.1: Sequences..... 404

7.2: Introduction to Lists

7.2: Introduction to Lists 404

The Repetition Operator 406

Iterating over a List with the for Loop 407

Indexing 408

The len Function 409

Using a for Loop to Iterate by Index Over a List 410

Lists Are Mutable..... 410

Concatenating Lists 412

7.2: (Noninteractive) Checkpoint Questions from the Book 413

7.3: List Slicing

7.3: List Slicing..... 414

7.3: (Noninteractive) Checkpoint Questions from the Book 416

7.4: Finding Items in Lists with the in Operator

7.4: Finding Items in Lists with the in Operator 417

7.4: (Noninteractive) Checkpoint Questions from the Book 418

7.5: List Methods and Useful Built-in Functions

7.5: List Methods and Useful Built-in Functions 418

7.5: (Noninteractive) Checkpoint Questions from the Book 425

7.6: Copying Lists

7.6: Copying Lists..... 425

7.7: Processing Lists

7.7: Processing Lists 427

In the Spotlight: Using List Elements in a Math Expression 427

Totaling the Values in a List 429

Averaging the Values in a List	429	8.2: String Slicing	487
Passing a List as an Argument to a Function	430	In the Spotlight: Extracting Characters from a String	489
Returning a List from a Function	431	8.2: (Noninteractive) Checkpoint Questions from the Book	490
In the Spotlight: Processing a List	433	8.3: Testing, Searching, and Manipulating Strings	
Randomly Selecting List Elements	435	8.3: Testing, Searching, and Manipulating Strings	491
Working with Lists and Files	436	String Methods	492
7.8: List Comprehensions		In the Spotlight: Validating the Characters in a Password	498
7.8: List Comprehensions	438	The Repetition Operator	500
Using if Clauses with List Comprehensions	440	Splitting a String	501
7.8: (Noninteractive) Checkpoint Questions from the Book	441	In the Spotlight: String Tokens	503
7.9: Two-Dimensional Lists		In the Spotlight: Reading CSV Files	505
7.9: Two-Dimensional Lists	441	8.3: (Noninteractive) Checkpoint Questions from the Book	506
7.9: (Noninteractive) Checkpoint Questions from the Book	446	Chapter 8: From the Book for Additional Practice	
7.10: Tuples		Chapter 8: Review Questions	507
7.10: Tuples	446	Chapter 8: Programming Exercises	510
Reassigning a Tuple to a Variable	448	9: Dictionaries and Sets	
Storing Mutable Objects in a Tuple	448	9: Topics	516
Converting Between Lists and Tuples	450	9.1: Dictionaries	516
7.10: (Noninteractive) Checkpoint Questions from the Book	451	9.1: Dictionaries	516
7.11: Plotting List Data with the matplotlib Package		Retrieving a Value from a Dictionary	517
7.11: Plotting List Data with the matplotlib Package	451	Using the in and not in Operators to Test for a Value in a Dictionary	518
Plotting a Line Graph	452	Adding Elements to an Existing Dictionary	519
Plotting a Bar Chart	463	Deleting Elements	519
Plotting a Pie Chart	468	Getting the Number of Elements in a Dictionary	520
7.11: (Noninteractive) Checkpoint Questions from the Book	471	Mixing Data Types in a Dictionary	521
Chapter 7: From the Book for Additional Practice		Creating an Empty Dictionary	522
Chapter 7: Review Questions	471	Using the for Loop to Iterate over a Dictionary ..	523
Chapter 7: Programming Exercises	475	Some Dictionary Methods	523
8: More About Strings		In the Spotlight: Using a Dictionary to Simulate a Deck of Cards	529
8: Topics	480	In the Spotlight: Storing Names and Birthdays in a Dictionary	532
8.1: Basic String Operations		The Dictionary Merge and Update Operators ...	537
8.1: Basic String Operations	480	Dictionary Comprehensions	538
Accessing the Individual Characters in a String ..	480		
String Concatenation	484		
Strings Are Immutable	486		
8.1: (Noninteractive) Checkpoint Questions from the Book	487		
8.2: String Slicing			

Using if Clauses with Dictionary Comprehensions	540	10.3: Working with Instances	593
9.1: (Noninteractive) Checkpoint Questions from the Book	541	In the Spotlight: Creating the CellPhone Class ..	596
9.2: Sets		Accessor and Mutator Methods	598
9.2: Sets	542	In the Spotlight: Storing Objects in a List	599
Adding and Removing Elements	544	Passing Objects as Arguments	600
Using the for Loop to Iterate over a Set	546	In the Spotlight: Pickling Your Own Objects	602
Using the in and not in Operators to Test for a Value in a Set	547	In the Spotlight: Storing Objects in a Dictionary	604
Finding the Union of Sets	547	10.3: (Noninteractive) Checkpoint Questions from the Book	612
Finding the Intersection of Sets	548	10.4: Techniques for Designing Classes	
Finding the Difference of Sets	549	10.4: Techniques for Designing Classes	612
Finding the Symmetric Difference of Sets	549	Finding the Classes in a Problem	613
Finding Subsets and Supersets	550	Identifying a Class's Responsibilities	620
In the Spotlight: Set Operations	551	10.4: (Noninteractive) Checkpoint Questions from the Book	625
Set Comprehensions	553	Chapter 10: From the Book for Additional Practice	
9.2: (Noninteractive) Checkpoint Questions from the Book	554	Chapter 10: Review Questions	625
9.3: Serializing Objects		Chapter 10: Programming Exercises	628
9.3: Serializing Objects	555	11: Inheritance	
9.3: (Noninteractive) Checkpoint Questions from the Book	561	11: Topics	633
Chapter 9: From the Book for Additional Practice		11.1: Introduction to Inheritance	
Chapter 9: Review Questions	561	11.1: Introduction to Inheritance	633
Chapter 9: Programming Exercises	568	In the Spotlight: Using Inheritance	643
10: Classes and Object-Oriented Programming		11.1: (Noninteractive) Checkpoint Questions from the Book	647
10: Topics	573	11.2: Polymorphism	
10.1: Procedural and Object-Oriented Programming		11.2: Polymorphism	647
10.1: Procedural and Object-Oriented Programming	573	11.2: (Noninteractive) Checkpoint Questions from the Book	654
10.1: (Noninteractive) Checkpoint Questions from the Book	576	Chapter 11: From the Book for Additional Practice	
10.2: Classes		Chapter 11: Review Questions	654
10.2: Classes	576	Chapter 11: Programming Exercises	656
Class Definitions	578	12: Recursion	
Hiding Attributes	583	12: Topics	658
Storing Classes in Modules	586	12.1: Introduction to Recursion	
The BankAccount Class	587	12.1: Introduction to Recursion	658
The __str__ Method	590	12.2: Problem Solving with Recursion	
10.2: (Noninteractive) Checkpoint Questions from the Book	593	12.2: Problem Solving with Recursion	660
10.3: Working with Instances		12.2: (Noninteractive) Checkpoint Questions from the Book	664

12.3: Examples of Recursive Algorithms	Adding Scrollbars to a Listbox	729
12.3: Examples of Recursive Algorithms.....	13.9: (Noninteractive) Checkpoint Questions from the Book	737
Chapter 12: From the Book for Additional Practice	13.10: Drawing Shapes with the Canvas Widget	
Chapter 12: Review Questions	13.10: Drawing Shapes with the Canvas Widget.....	737
Chapter 12: Programming Exercises	Drawing Lines: The create_line Method	740
13: GUI Programming	Drawing Rectangles: The create_rectangle Method	742
13: Topics.....	Drawing Ovals: The create_oval Method	745
13.1: Graphical User Interfaces	Drawing Arcs: The create_arc Method	747
13.1: Graphical User Interfaces.....	Drawing Polygons: The create_polygon Method	753
13.1: (Noninteractive) Checkpoint Questions from the Book	Drawing Text: The create_text Method	756
13.2: Using the tkinter Module	13.10: (Noninteractive) Checkpoint Questions from the Book	761
13.2: Using the tkinter Module.....	Chapter 13: From the Book for Additional Practice	
13.2: (Noninteractive) Checkpoint Questions from the Book	Chapter 13: Review Questions	761
13.3: Displaying Text with Label Widgets	Chapter 13: Programming Exercises	765
13.3: Displaying Text with Label Widgets	14: Database Programming	
13.3: (Noninteractive) Checkpoint Questions from the Book	14: Topics.....	769
13.4: Organizing Widgets with Frames	14.1: Database Management Systems	
13.4: Organizing Widgets with Frames.....	14.1: Database Management Systems.....	769
13.5: Button Widgets and Info Dialog Boxes	14.1: (Noninteractive) Checkpoint Questions from the Book	771
13.5: Button Widgets and Info Dialog Boxes	14.2: Tables, Rows, and Columns	
13.6: Getting Input with the Entry Widget	14.2: Tables, Rows, and Columns.....	771
13.6: Getting Input with the Entry Widget	Column Data Types	773
13.7: Using Labels as Output Fields	Primary Keys	774
13.7: Using Labels as Output Fields	Identity Columns	774
In the Spotlight: Creating a GUI Program	Allowing Null Values	776
13.7: (Noninteractive) Checkpoint Questions from the Book	14.2: (Noninteractive) Checkpoint Questions from the Book	776
13.8: Radio Buttons and Check Buttons	14.3: Opening and Closing a Database Connection with SQLite	
13.8: Radio Buttons and Check Buttons.....	14.3: Opening and Closing a Database Connection with SQLite.....	776
13.8: (Noninteractive) Checkpoint Questions from the Book	14.3: (Noninteractive) Checkpoint Questions from the Book	779
13.9: Listbox Widgets	14.4: Creating and Deleting Tables	
13.9: Listbox Widgets.....	14.4: Creating and Deleting Tables	779
Specifying the Size of the Listbox	Creating Multiple Tables	782
Using a Loop to Populate the Listbox	Creating a Table Only If It Does Not Already Exist	783
Selecting Items in a Listbox		
Deleting Items from a Listbox		
Executing a Callback Function When the User Clicks a Listbox Item		
In the Spotlight: The Time Zone Program		

Deleting a Table	783	14.9: Handling Database Exceptions	817
14.4: (Noninteractive) Checkpoint Questions from the Book	784	14.9: (Noninteractive) Checkpoint Questions from the Book	819
14.5: Adding Data to a Table		14.10: CRUD Operations	
14.5: Adding Data to a Table.....	784	14.10: CRUD Operations	819
Inserting Multiple Rows with One INSERT Statement	787	In the Spotlight: Inventory CRUD Application ..	819
Inserting NULL Data	787	14.11: Relational Data	
Inserting the Values of Variables	788	14.11: Relational Data	826
Watch Out for SQL Injection Attacks	790	Foreign Keys	827
14.5: (Noninteractive) Checkpoint Questions from the Book	790	Entity Relationship Diagrams	828
14.6: Querying Data with the SQL SELECT Statement		Creating Foreign Keys in SQL	829
14.6: Querying Data with the SQL SELECT Statement	791	Updating Relational Data	833
The SELECT Statement	792	Deleting Relational Data	833
Selecting All the Columns in a Table	795	Retrieving Columns from Multiple Tables in a SELECT Statement	834
Specifying Search Criteria with the WHERE Clause	797	In the Spotlight: A GUI Application to Read a Database	836
SQL Logical Operators: AND, OR, and NOT	799	14.11: (Noninteractive) Checkpoint Questions from the Book	841
String Comparisons in a SELECT Statement	800	Chapter 14: From the Book for Additional Practice	
Using the LIKE Operator	800	Chapter 14: Review Questions	841
Sorting the Results of a SELECT Query	802	Chapter 14: Programming Exercises	848
Aggregate Functions	804	Appendix A: Installing Python	
14.6: (Noninteractive) Checkpoint Questions from the Book	806	Appendix A: Installing Python.....	851
14.7: Updating and Deleting Existing Rows		Appendix B: Introduction to IDLE	
14.7: Updating and Deleting Existing Rows.....	807	Appendix B: Introduction to IDLE	853
Updating Multiple Columns	810	Appendix B: Introduction to IDLE	853
Determining the Number of Rows Updated	810	Writing a Python Program in the IDLE Editor....	855
Deleting Rows with the DELETE Statement	811	Color Coding.....	856
Determining the Number of Rows Deleted	813	Automatic Indentation	856
14.7: (Noninteractive) Checkpoint Questions from the Book	813	Saving a Program.....	857
14.7: (Noninteractive) Checkpoint Questions from the Book	813	Running a Program	857
14.8: More About Primary Keys		Appendix C: The ASCII Character Set	
14.8: More About Primary Keys.....	813	Appendix C: The ASCII Character Set.....	861
The RowID Column in SQLite	814	Appendix D: Predefined Named Colors	
Integer Primary Keys in SQLite	814	Appendix D: Predefined Named Colors	863
Primary Keys Other Than Integer	815	Appendix E: More About the import Statement	
Composite Keys	815	Appendix E: More About the import Statement	871
14.8: (Noninteractive) Checkpoint Questions from the Book	817		
14.9: Handling Database Exceptions			

Appendix F: Formatting Numeric Output with the format() Function

Appendix F: Formatting Numeric Output with the format() Function

Appendix F: Formatting Numeric Output with the format() Function.....	874
Formatting in Scientific Notation.....	875
Inserting Comma Separators.....	875
Specifying a Minimum Field Width.....	876
Formatting a Floating-Point Number as a Percentage.....	877
Formatting Integers.....	878

Appendix G: Installing Modules with the pip Utility

Appendix G: Installing Modules with the pip Utility.....	879
--	-----

Appendix H: Answers to Noninteractive Checkpoints

Appendix H: Answers to Noninteractive Checkpoints

Chapter 1: Answers to Noninteractive Checkpoints.....	880
Chapter 2: Answers to Noninteractive Checkpoints.....	880
Chapter 3: Answers to Noninteractive Checkpoints.....	883
Chapter 4: Answers to Noninteractive Checkpoints.....	885
Chapter 5: Answers to Noninteractive Checkpoints.....	886
Chapter 6: Answers to Noninteractive Checkpoints.....	888
Chapter 7: Answers to Noninteractive Checkpoints.....	889
Chapter 8: Answers to Noninteractive Checkpoints.....	891
Chapter 9: Answers to Noninteractive Checkpoints.....	892
Chapter 10: Answers to Noninteractive Checkpoints.....	894
Chapter 11: Answers to Noninteractive Checkpoints.....	895
Chapter 12: Answers to Noninteractive Checkpoints.....	895

Chapter 13: Answers to Noninteractive Checkpoints.....	895
Chapter 14: Answers to Noninteractive Checkpoints.....	897

Student Supplemental Materials

Student Supplemental Materials.....	900
-------------------------------------	-----

starting out with >>>

PYTHON®

SIXTH EDITION



TONY GADDIS

Dja65/123RF

STARTING OUT WITH PYTHON®

SIXTH EDITION

Tony Gaddis

Haywood Community College



Please contact <https://support.pearson.com/getsupport/s/contactsupport> with any queries on this content.

Director of Content Strategy: Dawn Murrin

Product Development: Tracy Johnson

Content Strategy Management: Tracy Johnson

Content Production: Abhijeet Gope, Bob Engelhardt, Sandra Rodriguez, Scott Disanno, Pallavi Pandit, Adarsh Sushanth

Product Management: Holly Stark
Marketing Manager: Krista Clark
Rights and Permissions: Anjali Singh
Cover Image: dja65/123RF

Copyright © 2023, 2021, 2018, 2015 by Pearson Education, Inc. or its affiliates, 221 River Street, Hoboken, NJ 07030. All Rights Reserved. Manufactured in the United States of America. This publication is protected by copyright, and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise. For information regarding permissions, request forms, and the appropriate contacts within the Pearson Education Global Rights and Permissions department, please visit www.pearsoned.com/permissions/.

Acknowledgments of third-party content appear on the appropriate page within the text.

PEARSON, ALWAYS LEARNING, and Revel are exclusive trademarks owned by Pearson Education, Inc. or its affiliates in the U.S. and/or other countries.

Unless otherwise indicated herein, any third-party trademarks, logos, or icons that may appear in this work are the property of their respective owners, and any references to third-party trademarks, logos, icons, or other trade dress are for demonstrative or descriptive purposes only. Such references are not intended to imply any sponsorship, endorsement, authorization, or promotion of Pearson's products by the owners of such marks, or any relationship between the owner and Pearson Education, Inc., or its affiliates, authors, licensees, or distributors.

Copyright © 2001-2022 Python Software Foundation; All Rights Reserved

Microsoft and/or its respective suppliers make no representations about the suitability of the information contained in the documents and related graphics published as part of the services for any purpose. All such documents and related graphics are provided "as is" without warranty of any kind. Microsoft and/or its respective suppliers hereby disclaim all warranties and conditions with regard to this information, including all warranties and conditions of merchantability, whether express, implied or statutory, fitness for a particular purpose, title and non-infringement. In no event shall Microsoft and/or its respective suppliers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of information available from the services.

The documents and related graphics contained herein could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Microsoft and/or its respective suppliers may make improvements and/or changes in the product(s) and/or the program(s) described herein at any time. Partial screen shots may be viewed in full within the software version specified.

Microsoft® and Windows® are registered trademarks of the Microsoft Corporation in the U.S.A. and other countries. This book is not sponsored or endorsed by or affiliated with the Microsoft Corporation.

Revel:

Revel for Starting Out with Python, Access Card 978-0-13-761913-9

Revel for Starting Out with Python, Instant Access 978-0-13-761915-3

Revel for Starting Out with Python, Combo Access Card 978-0-13-761914-6

Revel for Starting Out with Python, Inclusive Access 978-0-13-761918-4

Revel for Starting Out with Python, Print Offer [Loose-Leaf] 978-0-13-761910-8

Pearson+:

Pearson+ Starting Out with Python 978-0-13-787120-9

Starting Out with Python (Pearson eText Inclusive Access) 978-0-13-792251-2

Pearson's Commitment to Diversity, Equity, and Inclusion

Pearson is dedicated to creating bias-free content that reflects the diversity, depth, and breadth of all learners' lived experiences.

We embrace the many dimensions of diversity, including but not limited to race, ethnicity, gender, sex, sexual orientation, socioeconomic status, ability, age, and religious or political beliefs.

Our ambition is to purposefully contribute to a world where:

- Everyone has an equitable and lifelong opportunity to succeed through learning.
- Our educational content accurately reflects the histories and lived experiences of the learners we serve.
- Our educational products and services are inclusive and represent the rich diversity of learners.
- Our educational content prompts deeper discussions with students and motivates them to expand their own learning (and worldview).

Accessibility

We are also committed to providing products that are fully accessible to all learners. As per Pearson's guidelines for accessible educational Web media, we test and retest the capabilities of our products against the highest standards for every release, following the WCAG guidelines in developing new products for copyright year 2022 and beyond.

You can learn more about Pearson's commitment to accessibility at <https://www.pearson.com/us/accessibility.html>

Contact Us

While we work hard to present unbiased, fully accessible content, we want to hear from you about any concerns or needs with this Pearson product so that we can investigate and address them.

Please contact us with concerns about any potential bias at <https://www.pearson.com/report-bias.html>

For accessibility-related issues, such as using assistive technology with Pearson products, alternative text requests, or accessibility documentation, email the Pearson Disability Support team at disability.support@pearson.com

Contents in a Glance

Chapter 1: Introduction to Computers and Programming

Chapter 2: Input, Processing, and Output

Chapter 3: Decision Structures and Boolean Logic

Chapter 4: Repetition Structures

Chapter 5: Functions
Chapter 6: Files and Exceptions
Chapter 7: Lists and Tuples
Chapter 8: More About Strings
Chapter 9: Dictionaries and Sets
Chapter 10: Classes and Object-Oriented Programming
Chapter 11: Inheritance
Chapter 12: Recursion
Chapter 13: GUI Programming
Chapter 14: Database Programming
Appendix A: Installing Python
Appendix B: Introduction to IDLE
Appendix C: The ASCII Character Set
Appendix D: Predefined Named Colors
Appendix E: More About the import Statement
Appendix F: Formatting Numeric Output with the format() Function
Appendix G: Installing Modules with the pip Utility
Appendix H: Answers to Noninteractive Checkpoints
Student Supplemental Materials

Note: For detailed Table of Contents please see navigation bar or at the beginning of your Print Offer.

Location of VideoNotes

VideoNotes are narrated step-by-step video tutorials that show how to solve problems completely, from design through coding.

Chapter 1: Introduction to Computers and Programming

VideoNote 1-1: Using Interactive Mode in IDLE

VideoNote 1-2: Performing Exercise 2

Chapter 2: Input, Processing, and Output

VideoNote 2-1: Using the print function

VideoNote 2-2: Reading input from the Keyboard

VideoNote 2-3: Introduction to Turtle Graphics

VideoNote 2-4: The Sales Prediction Problem

Chapter 3: Decision Structures and Boolean Logic

VideoNote 3-1: The if Statement

VideoNote 3-2: The if-else Statement

VideoNote 3-3: The Areas of Rectangles Problem

Chapter 4: Repetition Structures

VideoNote 4-1: The while Loop

VideoNote 4-2: The for Loop

VideoNote 4-3: Bug Collector Problem

Chapter 5: Functions

VideoNote 5-1: Defining and Calling a Function

VideoNote 5-2: Passing Arguments to a Function

VideoNote 5-3: Writing a Value-Returning Function

VideoNote 5-4: The Kilometer Converter Problem

VideoNote 5-5: The Feet to Inches Problem

Chapter 6: Files and Exceptions

VideoNote 6-1: Using Loops to Process Files

VideoNote 6-2: File Display

Chapter 7: Lists and Tuples

VideoNote 7-1: List Slicing

VideoNote 7-2: Lottery Number Generator Problem

Chapter 8: More About Strings

VideoNote 8-1: Vowels and Consonants Problem

Chapter 9: Dictionaries and Sets

VideoNote 9-1: Introduction to Dictionaries

VideoNote 9-2: Introduction to Sets

VideoNote 9-3: The Capital Quiz Problem

Chapter 10: Classes and Object-Oriented Programming

VideoNote 10-1: Classes and Objects

VideoNote 10-2: The Pet Class

Chapter 11: Inheritance

VideoNote 11-1: Person and Customer Classes

Chapter 12: Recursion

VideoNote 12-1: Recursive Multiplication

Chapter 13: GUI Programming

VideoNote 13-1: Creating a Simple GUI

VideoNote 13-2: Responding to Button Clicks

VideoNote 13-3: Name and Address Problem

Chapter 14: Database Programming

VideoNote 14-1: Opening and Closing a Database Connection

VideoNote 14-2: Creating a Table

VideoNote 14-3: Adding Data to a Table

VideoNote 14-4: The SELECT Statement

VideoNote 14-5: Updating Rows

VideoNote 14-6: Getting Started with the Population Database Problem

Appendix B: Introduction to IDLE

VideoNote B-1: Intro to IDLE

Preface

Welcome to *Starting Out with Python*, Sixth Edition. This book uses the Python language to teach programming concepts and problem-solving skills, without assuming any previous programming experience. With easy-to-understand examples, pseudocode, flowcharts, and other tools, the student learns how to design the logic of programs then implement those programs using Python. This book is ideal for an introductory programming course or a programming logic and design course using Python as the language.

As with all the books in the *Starting Out With* series, the hallmark of this text is its clear, friendly, and easy-to-understand writing. In addition, it is rich in example programs that are concise and practical. The programs in this book include short examples that highlight specific programming topics, as well as more involved examples that focus on problem solving. Each chapter provides one or more case studies that provide step-by-step analysis of a specific problem and shows the student how to solve it.

Control Structures First, Then Classes

Python is a fully object-oriented programming language, but students do not have to understand object-oriented concepts to start programming in Python. This text first introduces the student to the fundamentals of data storage, input and output, control structures, functions, sequences and lists, file I/O, and objects that are created from standard library classes. Then the student learns to write classes, explores the topics of inheritance and polymorphism, and learns to write recursive functions. Finally, the student learns to develop simple event-driven GUI applications.

Changes in the Sixth Edition

This book's clear writing style remains the same as in the previous edition. However, many additions and improvements have been made, which are summarized here:

- **Updated for Python 3.9** – This edition uses new language features from versions of Python up through Python 3.9.
- **The with Statement -- Chapter 6** now introduces the `with` statement as a way to open files. Many examples of using the `with` statement with files have been added throughout the book.
- **Multiple Assignment** -- This edition introduces multiple assignment in **Chapter 2**.
- **Single-Line if Statements** -- A new section on single-line `if` statements has been added to **Chapter 3**.
- **Conditional Expressions** -- **Chapter 3** now introduces conditional expressions and the ternary operator.
- **Walrus Operator and Assignment Expressions** -- A new section on the walrus operator and assignment expressions has been added to **Chapter 3**. **Chapter 3** shows an example using an assignment expression in an `if` statement, and **Chapter 4** shows an example of using an assignment expression in a `while` loop.

- **Using the `while` Loop as a Count-Controlled Loop -- Chapter 4** has a new section on counter variables and using the `while` statement to write count-controlled loops.
- **Single-Line `while` Loops --** A new section on single-line `while` Loops has been added to **Chapter 4**.
- **Using `break`, `continue`, and `else` With Loops -- Chapter 4** now has a section on using `break` and `continue` with loops, and Python's unique `else` clause with loops.
- **Keyword-Only Parameters -- Chapter 5** now discusses keyword-only parameters and how to implement them in a function.
- **Positional-Only Parameters -- Chapter 5** now discusses positional-only parameters and how to implement them in a function.
- **Default Arguments -- Chapter 5** now has a section on default function arguments.
- **Using `count` and `sum` with Lists --** A discussion of the `count` method and the `sum` function with lists has been added to **Chapter 7**.
- **Storing Mutable Objects in a Tuple --** A new section has been added to **Chapter 7** that discusses the immutability of tuples, and how mutable objects can be stored in tuples.
- **Dictionary Merge and Update Operators -- Chapter 9** has a new section that discusses the dictionary's merge and update operators.

Brief Overview of Each Chapter

Chapter 1: Introduction to Computers and Programming

This chapter begins by giving a very concrete and easy-to-understand explanation of how computers work, how data is stored and manipulated, and why we write programs in highlevel languages. An introduction to Python, interactive mode, script mode, and the IDLE environment are also given.

Chapter 2: Input, Processing, and Output

This chapter introduces the program development cycle, variables, data types, and simple programs that are written as sequence structures. The student learns to write simple programs that read input from the keyboard, perform mathematical operations, and produce formatted screen output. Pseudocode and flowcharts are also introduced as tools for designing programs. The chapter also includes an optional introduction to the turtle graphics library.

Chapter 3: Decision Structures and Boolean Logic

In this chapter, the student learns about relational operators and Boolean expressions and is shown how to control the flow of a program with decision structures. The `if`, `if-else`, and `if-elif-else` statements are covered. Nested decision structures and logical operators are discussed as well. The chapter also includes an optional turtle graphics section, with a discussion of how to use decision structures to test the state of the turtle.

Chapter 4: Repetition Structures

This chapter shows the student how to create repetition structures using the `while` loop and `for` loop. Counters, accumulators, running totals, and sentinels are discussed, as well as techniques for writing input validation loops. The chapter also includes an optional section on using loops to draw designs with the turtle graphics library.

Chapter 5: Functions

In this chapter, the student first learns how to write and call void functions. The chapter shows the benefits of using functions to modularize programs and discusses the top-down design approach. Then, the student learns to pass arguments to functions. Common library functions, such as those for generating random numbers, are discussed. After learning how to call library functions and use their return value, the student learns to define and call their own functions. Then the student learns how to use modules to organize functions. An optional section includes a discussion of modularizing turtle graphics code with functions.

Chapter 6: Files and Exceptions

This chapter introduces sequential file input and output. The student learns to read and write large sets of data and store data as fields and records. The chapter concludes by discussing exceptions and shows the student how to write exception-handling code.

Chapter 7: Lists and Tuples

This chapter introduces the student to the concept of a sequence in Python and explores the use of two common Python sequences: lists and tuples. The student learns to use lists for arraylike operations, such as storing objects in a list, iterating over a list, searching for items in a list, and calculating the sum and average of items in a list. The chapter discusses list comprehension expressions, slicing, and many of the list methods. One- and two-dimensional lists are covered. The chapter also includes a discussion of the `matplotlib` package, and how to use it to plot charts and graphs from lists.

Chapter 8: More About Strings

In this chapter, the student learns to process strings at a detailed level. String slicing and algorithms that step through the individual characters in a string are discussed, and several built-in functions and string methods for character and text processing are introduced. This chapter also includes examples of string tokenizing and parsing CSV files.

Chapter 9: Dictionaries and Sets

This chapter introduces the dictionary and set data structures. The student learns to store data as key-value pairs in dictionaries, search for values, change existing values, add new key-value pairs, delete key-value pairs, and write dictionary comprehensions. The student learns to store values as unique elements in sets and perform common set operations such as union, intersection, difference, and symmetric difference. Set comprehensions are also introduced. The chapter concludes with a discussion of object serialization and introduces the student to the Python `pickle` module.

Chapter 10: Classes and Object-Oriented Programming

This chapter compares procedural and object-oriented programming practices. It covers the fundamental concepts of classes and objects. Attributes, methods, encapsulation and data hiding, `__init__` functions (which are similar to constructors), accessors, and mutators are discussed. The student learns how to model classes with UML and how to find the classes in a particular problem.

Chapter 11: Inheritance

The study of classes continues in this chapter with the subjects of inheritance and polymorphism. The topics covered include superclasses, subclasses, how `__init__` functions work in inheritance, method overriding, and polymorphism.

Chapter 12: Recursion

This chapter discusses recursion and its use in problem solving. A visual trace of recursive calls is provided, and recursive applications are discussed. Recursive algorithms for many tasks are presented, such as finding factorials, finding a greatest common denominator (GCD), and summing a range of values in a list, and the classic Towers of Hanoi example are presented.

Chapter 13: GUI Programming

This chapter discusses the basic aspects of designing a GUI application using the `tkinter` module in Python. Fundamental widgets, such as labels, buttons, entry fields, radio buttons, check buttons, list boxes, and dialog boxes, are covered. The student also learns how events work in a GUI application and how to write callback functions to handle events. The Chapter includes a discussion of the Canvas widget, and how to use it to draw lines, rectangles, ovals, arcs, polygons, and text.

Chapter 14: Database Programming

This chapter introduces the student to database programming. The chapter first introduces the basic concepts of databases, such as tables, rows, and primary keys. Then the student learns to use SQLite to connect to a database in Python. SQL is introduced and the student learns to execute queries and statements that search for rows, add new rows, update existing rows, and delete rows. CRUD applications are demonstrated, and the chapter concludes with a discussion of relational data.

Appendix A: Installing Python

This appendix explains how to download and install the latest Python distribution.

Appendix B: Introduction to IDLE

This appendix gives an overview of the IDLE integrated development environment that comes with Python.

Appendix C: The ASCII Character Set

As a reference, this appendix lists the ASCII character set.

Appendix D: Predefined Named Colors

This appendix lists the predefined color names that can be used with the turtle graphics library, `matplotlib` and `tkinter`.

Appendix E: More About the `import` Statement

This appendix discusses various ways to use the `import` statement. For example, you can use the `import` statement to import a module, a class, a function, or to assign an alias to a module.

Appendix F: Formatting Numeric Output with the `format()` Function

This appendix discusses the `format()` function and shows how to use its format specifiers to control the way that numeric values are displayed.

Appendix G: Installing Modules with the `pip` Utility

This appendix discusses how to use the `pip` utility to install third-party modules from the Python Package Index, or PyPI.

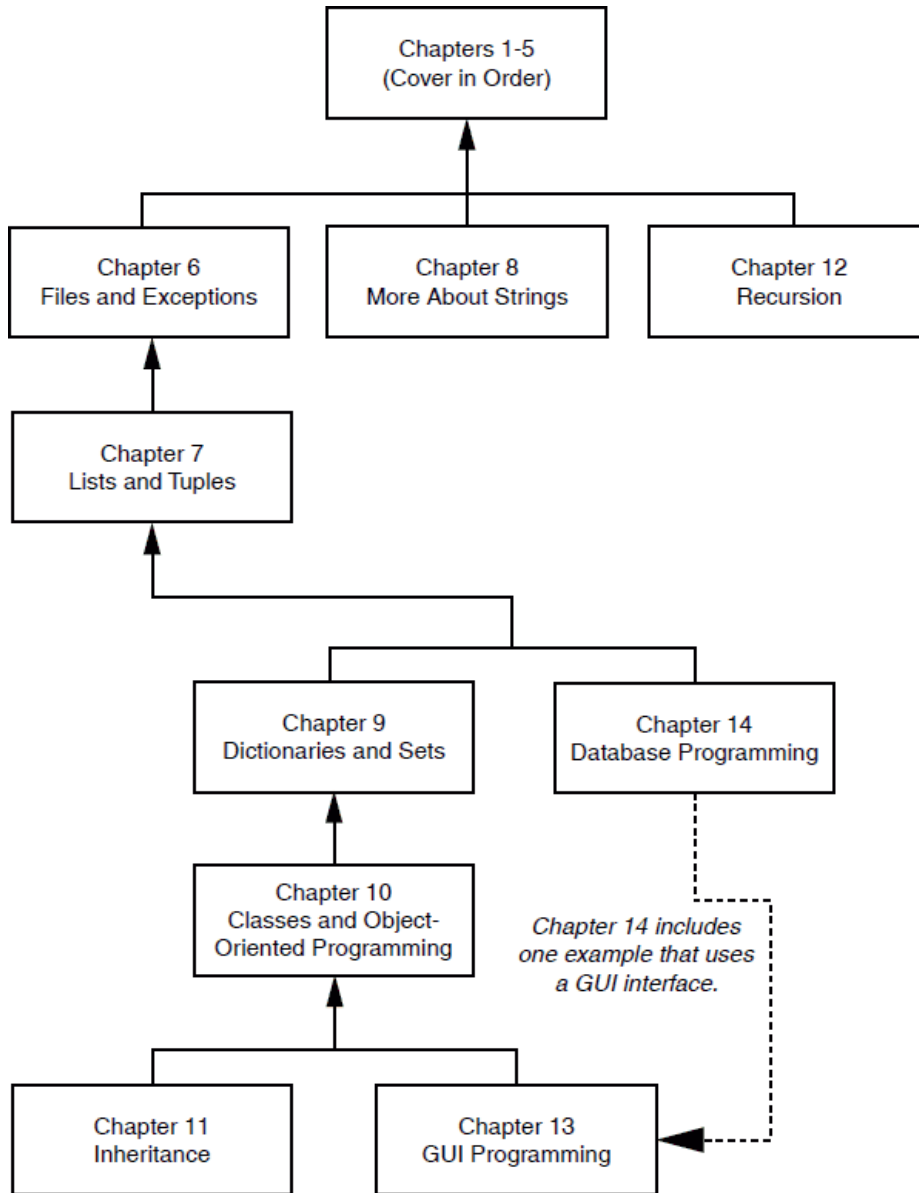
Appendix H: Answers to Checkpoints

This appendix gives the answers to the Checkpoint questions that appear throughout the text.

Organization of the Text

The text teaches programming in a step-by-step manner. Each chapter covers a major set of topics and builds knowledge as students progress through the book. Although the chapters can be easily taught in their existing sequence, you do have some flexibility in the order that you wish to cover them. **Figure P-1** shows chapter dependencies. Each box represents a chapter or a group of chapters. An arrow points from a chapter to the chapter that must be covered before it.

Figure P-1
Chapter dependencies



Features of the Text

Concept

Each major section of the text starts with a concept statement.

Statements

This statement concisely summarizes the main point of the section.

Example Programs

Each chapter has an abundant number of complete and partial example programs, each designed to highlight the current topic.

In the Spotlight

Each chapter has one or more *In the Spotlight* case studies that Case Studies provide detailed, step-by-step analysis of problems and show the student how to solve them.

VideoNotes

VideoNotes are narrated step-by-step video tutorials that show how to solve problems completely, from design through coding.

Notes

Notes appear at several places throughout the text. They are short explanations of interesting or often misunderstood points relevant to the topic at hand.

Tips

Tips advise the student on the best techniques for approaching different programming problems.

Warnings

Warnings caution students about programming techniques or practices that can lead to malfunctioning programs or lost data.

Checkpoints

(Noninteractive) Checkpoint Questions from the Book are questions placed at intervals throughout each chapter. They are designed to query the student's knowledge quickly after learning a new topic. Please note that the Checkpoint exercises in this textbook may differ from the Checkpoint questions embedded in the digital product Revel Starting Out with Python.

Review Questions

Each chapter presents a thorough and diverse set of review questions and exercises. They include Multiple Choice, True/False, Algorithm Workbench, and Short Answer.

Programming Exercises

Each chapter offers a pool of programming exercises designed to solidify the student's knowledge of the topics currently being studied. Please note that the end-of-chapter Programming Exercises in this textbook may differ from the end-of-chapter Programming Exercises embedded in the digital product Revel Starting Out with Python.

Supplements

Student Online Resources

Many student resources are available for this book from the publisher. The following items are available at https://media.pearsoncmg.com/ph/esm/ecs_gaddis_sowpython_6e/cw/

Instructor Resources

The following supplements are available to qualified instructors only:

- Solutions for the Section Quizzes, Chapter Quizzes, Programming Projects, Checkpoint self-review items, and Live Code Example practice activities are contained in the Revel Solutions Manual
- Solutions for the review questions and exercises

- PowerPoint presentation slides for each chapter
- Test bank
- TestGen
- Errata

Visit the Pearson Education Instructor Resource Center (www.pearsonhighered.com/irc) or contact your local Pearson Education campus representative for information on how to access them.

Acknowledgments

I would like to thank the following faculty reviewers for their insight, expertise, and thoughtful recommendations:

Desmond K. H. Chun	Raymond Pettit
<i>Chabot Community College</i>	<i>Abilene Christian University</i>
Sonya Dennis	Janet Renwick
<i>Morehouse College</i>	<i>University of Arkansas–Fort Smith</i>
Barbara Goldner	Haris Ribic
<i>North Seattle Community College</i>	<i>SUNY at Binghamton</i>
Paul Gruhn	Ken Robol
<i>Manchester Community College</i>	<i>Beaufort Community College</i>
Bob Husson	Eric Shaffer
<i>Craven Community College</i>	<i>University of Illinois at Urbana-Champaign</i>
Diane Innes	Tom Stokke
<i>Sandhills Community College</i>	<i>University of North Dakota</i>
Daniel Jinguji	Anita Sutton
<i>North Seattle Community College</i>	<i>Germanna Community College</i>
John Kinuthia	Ann Ford Tyson
<i>Nazareth College of Rochester</i>	<i>Florida State University</i>

Frank Liu	Karen Ughetta
<i>Sam Houston State University</i>	<i>Virginia Western Community College</i>
Gary Marrer	Christopher Urban
<i>Glendale Community College</i>	<i>SUNY Institute of Technology</i>
Keith Mehl	Nanette Veilleux
<i>Chabot College</i>	<i>Simmons College</i>
Shyamal Mitra	Brent Wilson
<i>University of Texas at Austin</i>	<i>George Fox University</i>
Vince Offenback	Linda F. Wilson
<i>North Seattle Community College</i>	<i>Texas Lutheran University</i>
Smiljana Petrovic	
<i>Iona College</i>	

I would like to thank the faculty, staff, and administration at Haywood Community College for the opportunity to build a career teaching the subjects that I love. I would also like to thank my family and friends for their support in all of my projects.

It is a great honor to be published by Pearson, and I am extremely fortunate to have Tracy Johnson as my Editor and Content Manager. She and her colleagues Holly Stark, Erin Sullivan, Krista Clark, Scott Disanno, Sandra Rodriguez, Bob Engelhardt, Abhijeet Gope, Adarsh Sushanth, Pallavi Pandit, and Anu Sivakolundu have worked tirelessly to produce and promote this book. Thanks to you all!

About the Author



Tony Gaddis

Tony Gaddis's career includes twenty years as a faculty member at Haywood Community College. He has taught computer science courses, both in-person and online, to a wide variety of students. He particularly enjoys the challenges of teaching non-majors and others who initially struggle with the concepts of programming. He is an award-winning instructor who was previously selected as the NC Community College Teacher of the Year.

Tony is also a prolific author. His Starting Out With... series of textbooks, published by Pearson, covers a wide range of programming languages and pedagogical approaches for the CS1 and CS2 classrooms. Each of Tony's books encapsulates his passion for teaching and his experience explaining difficult concepts to beginning students.

Ordering Options

Revel

For more information about the Revel for this title, please visit: <https://www.pearsonhighered.com/revel/educators/features/>

Enhanced Revel Features: <https://www.pearson.com/au/revel/educators/enhanced-features/index.html>

Revel Instructor Help: <https://help.pearsoncmg.com/glp/en-us/pearson/instructor/Content/index.htm>

Pearson+

For more information about Pearson+ for this title, please visit <https://www.pearson.com/en-us/pearsonplus.html> and <https://plc.pearson.com/en-US/our-products-and-services/pearson-plus>

Pearson+ Help Center: <https://www.pearson.com/en-us/pearsonplus/support.html>

Chapter 1

Introduction to Computers and Programming

Topics

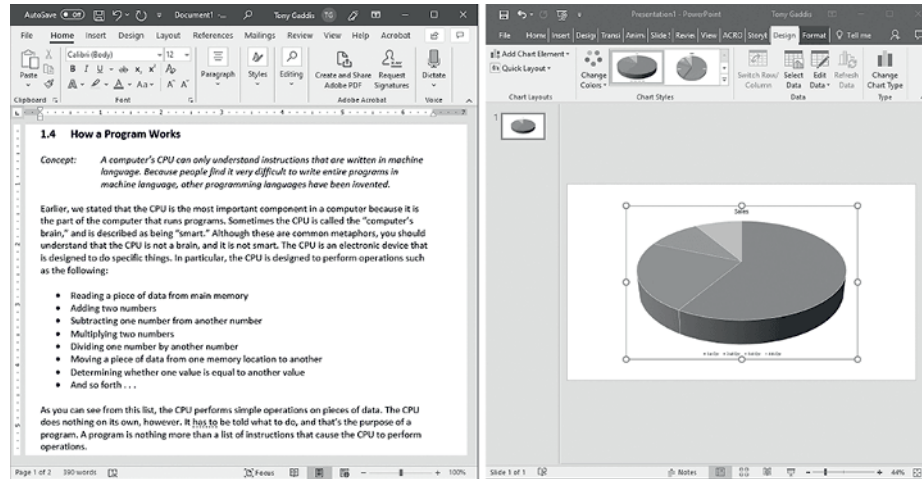
- 1.1 Introduction
- 1.2 Hardware and Software
- 1.3 How Computers Store Data
- 1.4 How a Program Works
- 1.5 Using Python

1.1: Introduction

Think about some of the different ways that people use computers. In school, students use computers for tasks such as writing papers, searching for articles, sending email, and participating in online classes. At work, people use computers to analyze data, make presentations, conduct business transactions, communicate with customers and coworkers, control machines in manufacturing facilities, and do many other things. At home, people use computers for tasks such as paying bills, shopping online, communicating with friends and family, and playing games. And don't forget that cell phones, tablets, smart phones, car navigation systems, and many other devices are computers too. The uses of computers are almost limitless in our everyday lives.

Computers can perform such a wide variety of tasks because they can be programmed. This means that computers are not designed to do just one job, but to do any job that their programs tell them to do. A *program* is a set of instructions that a computer follows to perform a task. For example, **Figure 1-1** shows screens using Microsoft Word and PowerPoint, two commonly used programs.

Figure 1-1
A word processing program and a presentation program



Used with permission from Microsoft.

Programs are commonly referred to as *software*. Software is essential to a computer because it controls everything the computer does. All of the software that we use to make our computers useful is created by individuals working as programmers or software developers. A *programmer*, or *software developer*, is a person with the training and skills necessary to design, create, and test computer programs. Computer programming is an exciting and rewarding career. Today, you will find programmers' work used in business, medicine, government, law enforcement, agriculture, academics, entertainment, and many other fields.

This book introduces you to the fundamental concepts of computer programming using the Python language. The Python language is a good choice for beginners because it is easy to learn and programs can be written quickly using it. Python is also a powerful language, popular with professional software developers. In fact, it has been reported that Python is used by Google, NASA, YouTube, various game companies, the New York Stock Exchange, and many others.

Before we begin exploring the concepts of programming, you need to understand a few basic things about computers and how they work. This chapter will build a solid foundation of knowledge that you will continually rely on as you study computer science. First, we will discuss the physical components of which computers are commonly made. Next, we will look at how computers store data and execute programs. Finally, you will get a quick introduction to the software that you will use to write Python programs.

1.2: Hardware and Software

CONCEPT: The physical devices of which a computer is made are referred to as the computer's hardware. The programs that run on a computer are referred to as software.

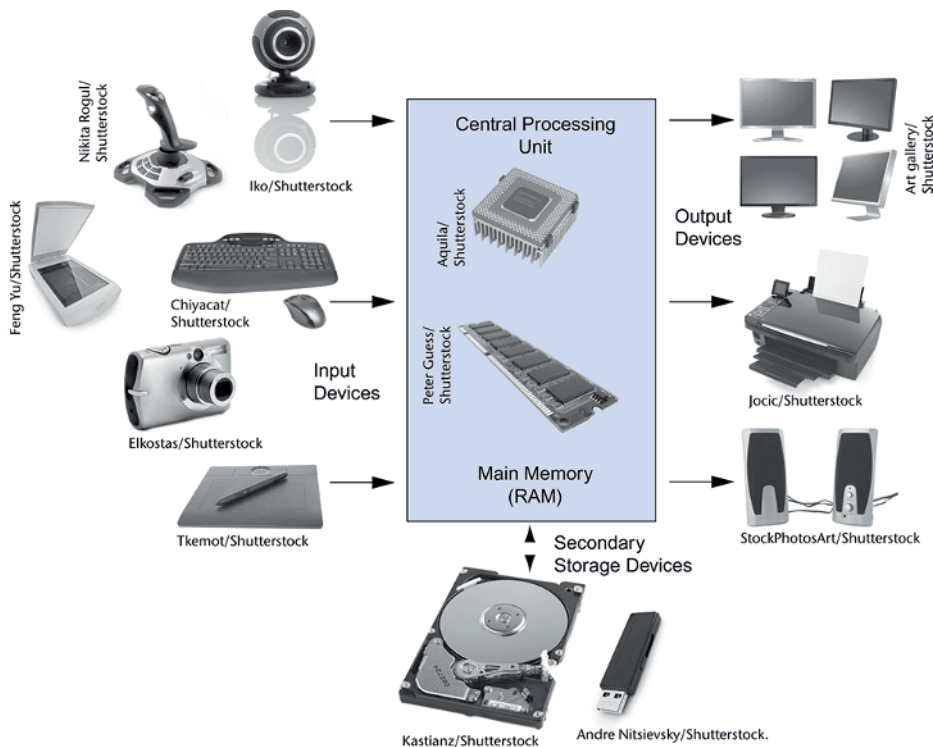
Hardware

The term *hardware* refers to all of the physical devices, or *components*, of which a computer is made. A computer is not one single device, but a system of devices that all work together. Like the different instruments in a symphony orchestra, each device in a computer plays its own part.

If you have ever shopped for a computer, you've probably seen sales literature listing components such as microprocessors, memory, disk drives, video displays, graphics cards, and so on. Unless you already know a lot about computers, or at least have a friend that does, understanding what these different components do might be challenging. As shown in **Figure 1-2**, a typical computer system consists of the following major components:

- The central processing unit (CPU)
- Main memory
- Secondary storage devices
- Input devices
- Output devices

Figure 1-2
Typical components of a computer system



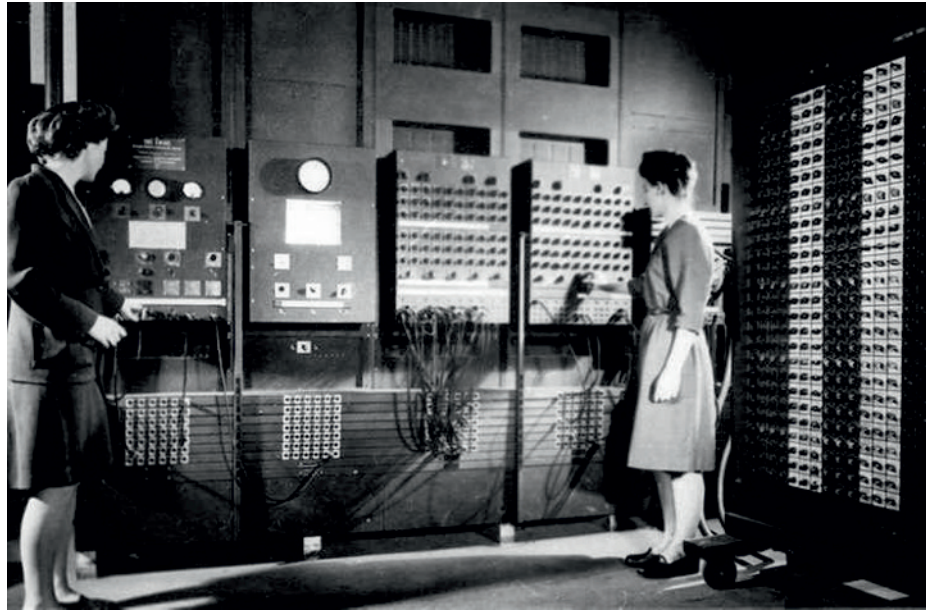
Let's take a closer look at each of these components.

The CPU

When a computer is performing the tasks that a program tells it to do, we say that the computer is *running* or *executing* the program. The *central processing unit*, or *CPU*, is the part of a computer that actually runs programs. The CPU is the most important component in a computer because without it, the computer could not run software.

In the earliest computers, CPUs were huge devices made of electrical and mechanical components such as vacuum tubes and switches. **Figure 1-3** shows such a device. The two women in the photo are working with the historic ENIAC computer. The **ENIAC**, which is considered by many to be the world's first programmable electronic computer, was built in 1945 to calculate artillery ballistic tables for the U.S. Army. This machine, which was primarily one big CPU, was 8 feet tall, 100 feet long, and weighed 30 tons.

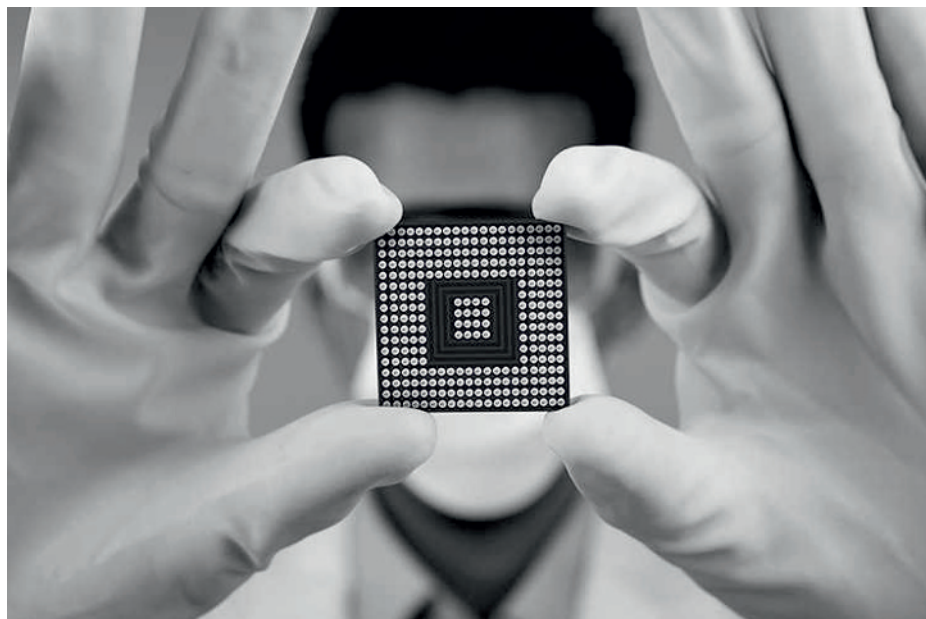
Figure 1-3
The ENIAC computer



Courtesy of U.S. Army Historic Computer Images

Today, CPUs are small chips known as *microprocessors*. **Figure 1-4** shows a photo of a lab technician holding a modern microprocessor. In addition to being much smaller than the old electromechanical CPUs in early computers, microprocessors are also much more powerful.

Figure 1-4
A lab technician holds a modern microprocessor



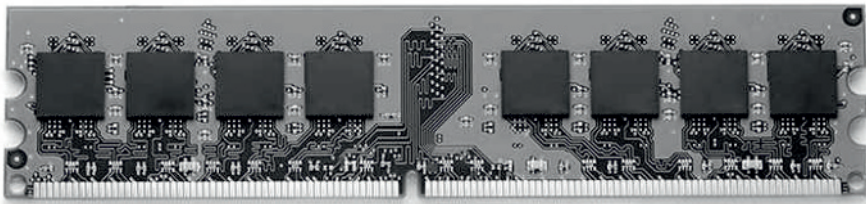
Creativa Images/Shutterstock

Main Memory

You can think of *main memory* as the computer's work area. This is where the computer stores a program while the program is running, as well as the data that the program is working with. For example, suppose you are using a word processing program to write an essay for one of your classes. While you do this, both the word processing program and the essay are stored in main memory.

Main memory is commonly known as *random-access memory*, or *RAM*. It is called this because the CPU is able to quickly access data stored at any random location in RAM. RAM is usually a *volatile* type of memory that is used only for temporary storage while a program is running. When the computer is turned off, the contents of RAM are erased. Inside your computer, RAM is stored in chips, similar to the ones shown in **Figure 1-5**.

Figure 1-5
Memory chips



Garsya/Shutterstock

Secondary Storage Devices

Secondary storage is a type of memory that can hold data for long periods of time, even when there is no power to the computer. Programs are normally stored in secondary memory and loaded into main memory as needed. Important data, such as word processing documents, payroll data, and inventory records, is saved to secondary storage as well.

The most common type of secondary storage device is the *disk drive*. A traditional disk drive stores data by magnetically encoding it onto a spinning circular disk. *Solid-state drives*, which store data in solid-state memory, are increasingly becoming popular. A solid-state drive has no moving parts and operates faster than a traditional disk drive. Most computers have some sort of secondary storage device, either a traditional disk drive or a solid-state drive, mounted inside their case. External storage devices, which connect to one of the computer's communication ports, are also available. External storage devices can be used to create backup copies of important data or to move data to another computer.

In addition to external storage devices, many types of devices have been created for copying data and for moving it to other computers. For example, *USB drives* are small devices that plug into the computer's USB (universal serial bus) port and appear to the system as a disk drive. These drives do not actually contain a disk, however. They store data in a special type of memory known as *flash memory*. USB drives, which are also known as *memory sticks* and *flash drives*, are inexpensive, reliable, and small enough to be carried in your pocket.

Input Devices

Input is any data the computer collects from people and from other devices. The component that collects the data and sends it to the computer is called an *input device*. Common input devices are the keyboard, mouse, touchscreen, scanner, microphone, and digital camera. Disk drives and optical drives can also be considered input devices, because programs and data are retrieved from them and loaded into the computer's memory.

Output Devices

Output is any data the computer produces for people or for other devices. It might be a sales report, a list of names, or a graphic image. The data is sent to an *output device*, which formats and presents it. Common output devices are video displays and printers. Disk drives can also be considered output devices because the system sends data to them in order to be saved.

Software

If a computer is to function, software is not optional. Everything computer does, from the time you turn the power switch on until you shut the system down, is under the control of software. There are two general categories of software: system software and application software. Most computer programs clearly fit into one of these two categories. Let's take a closer look at each.

System Software

The programs that control and manage the basic operations of a computer are generally referred to as *system software*. System software typically includes the following types of programs:

Operating Systems An *operating system* is the most fundamental set of programs on a computer. The operating system controls the internal operations of the computer's hardware, manages all of the devices connected to the computer, allows data to be saved to and retrieved from storage devices, and allows other programs to run on the computer. Popular operating systems for laptop and desktop computers include Windows, macOS, and Linux. Popular operating systems for mobile devices include Android and iOS.

Utility Programs A *utility program* performs a specialized task that enhances the computer's operation or safeguards data. Examples of utility programs are virus scanners, file compression programs, and data backup programs.

Software Development Tools *Software development tools* are the programs that programmers use to create, modify, and test software. Assemblers, compilers, and interpreters are examples of programs that fall into this category.

Application Software

Programs that make a computer useful for everyday tasks are known as *application software*. These are the programs that people normally spend most of their time running on their computers. **Figure 1-1**, at the beginning of this chapter, shows screens from two commonly used applications: Microsoft Word, a word processing program, and PowerPoint, a presentation program. Some other examples of application software are spreadsheet programs, email programs, web browsers, and game programs.

1.2: (Noninteractive) Checkpoint Questions from the Book

- 1.1 What is a program?
- 1.2 What is hardware?
- 1.3 List the five major components of a computer system.
- 1.4 What part of the computer actually runs programs?
- 1.5 What part of the computer serves as a work area to store a program and its data while the program is running?
- 1.6 What part of the computer holds data for long periods of time, even when there is no power to the computer?
- 1.7 What part of the computer collects data from people and from other devices?
- 1.8 What part of the computer formats and presents data for people or other devices?
- 1.9 What fundamental set of programs control the internal operations of the computer's hardware?
- 1.10 What do you call a program that performs a specialized task, such as a virus scanner, a file compression program, or a data backup program?
- 1.11 Word processing programs, spreadsheet programs, email programs, web browsers, and game programs belong to what category of software?

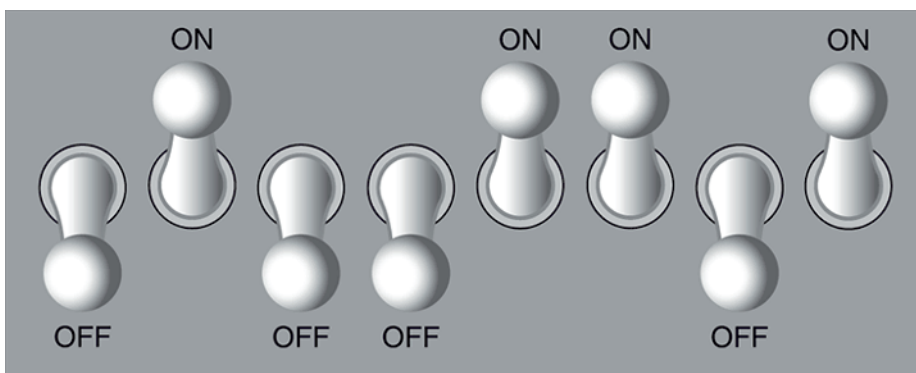
1.3: How Computers Store Data

CONCEPT: All data that is stored in a computer is converted to sequences of 0s and 1s.

A computer's memory is divided into tiny storage locations known as *bytes*. One byte is only enough memory to store a letter of the alphabet or a small number. In order to do anything meaningful, a computer has to have lots of bytes. Most computers today have millions, or even billions, of bytes of memory.

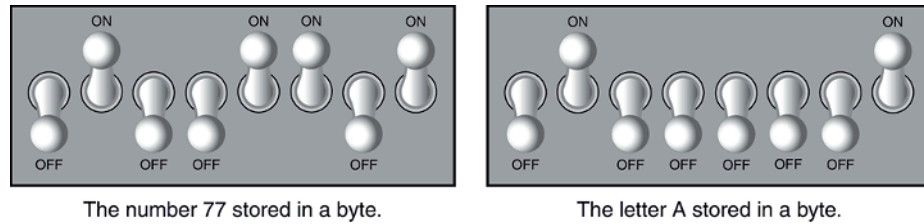
Each byte is divided into eight smaller storage locations known as bits. The term *bit* stands for *binary digit*. Computer scientists usually think of bits as tiny switches that can be either on or off. Bits aren't actual "switches," however, at least not in the conventional sense. In most computer systems, bits are tiny electrical components that can hold either a positive or a negative charge. Computer scientists think of a positive charge as a switch in the *on* position, and a negative charge as a switch in the *off* position. **Figure 1-6** shows the way that a computer scientist might think of a byte of memory: as a collection of switches that are each flipped to either the on or off position.

Figure 1-6
Think of a byte as eight switches



When a piece of data is stored in a byte, the computer sets the eight bits to an on/off pattern that represents the data. For example, the pattern on the left in **Figure 1-7** shows how the number 77 would be stored in a byte, and the pattern on the right shows how the letter A would be stored in a byte. We explain below how these patterns are determined.

Figure 1-7
Bit patterns for the number 77 and the letter A



Storing Numbers

A bit can be used in a very limited way to represent numbers. Depending on whether the bit is turned on or off, it can represent one of two different values. In computer systems, a bit that is turned off represents the number 0, and a bit that is turned on represents the number 1. This corresponds perfectly to the *binary numbering system*. In the binary numbering system (or *binary*, as it is usually called), all numeric values are written as sequences of 0s and 1s. Here is an example of a number that is written in binary:

```
10011101
```

The position of each digit in a binary number has a value assigned to it. Starting with the rightmost digit and moving left, the position values are 2^0 , 2^1 , 2^2 , 2^3 , and so forth, as shown in **Figure 1-8**. **Figure 1-9** shows the same diagram with the position values calculated. Starting with the rightmost digit and moving left, the position values are 1, 2, 4, 8, and so forth.

Figure 1-8
The values of binary digits as powers of 2

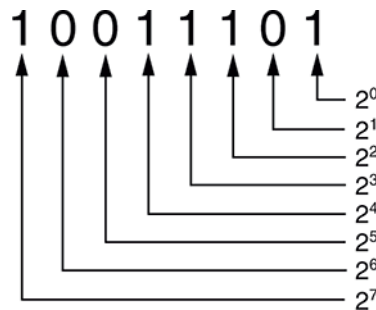
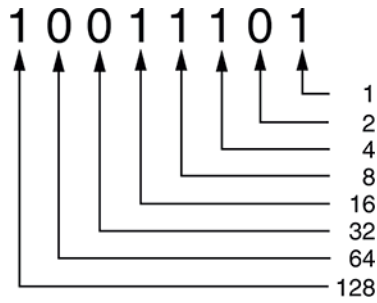


Figure 1-9
The values of binary digits



To determine the value of a binary number, you simply add up the position values of all the 1s. For example, in the binary number 10011101, the position values of the 1s are 1, 4, 8, 16, and 128. This is shown in **Figure 1-10**. The sum of all of these position values is 157. So, the value of the binary number 10011101 is 157.

Figure 1-10
Determining the value of 10011101

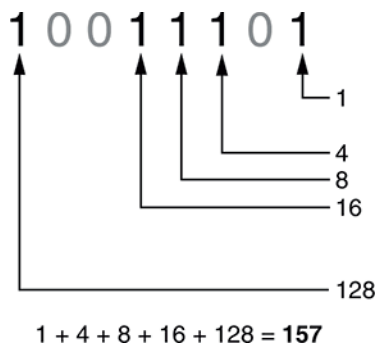
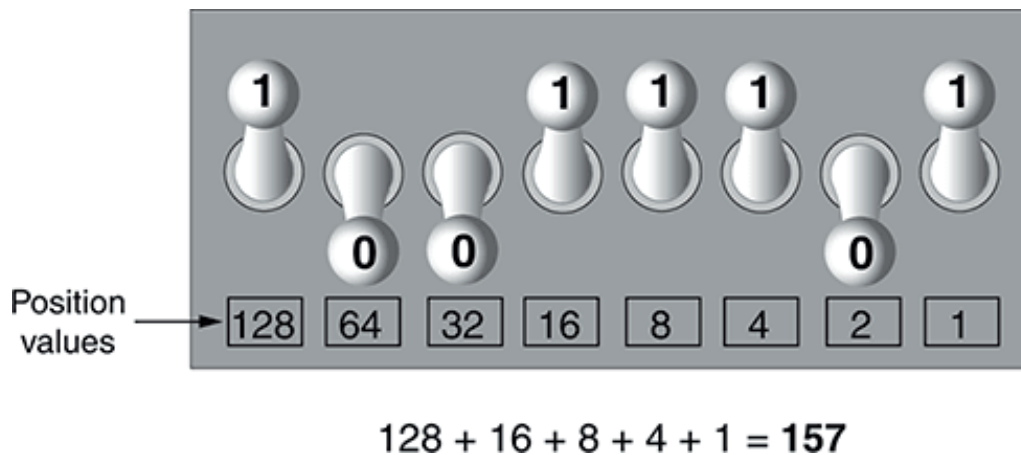


Figure 1-11 shows how you can picture the number 157 stored in a byte of memory. Each 1 is represented by a bit in the on position, and each 0 is represented by a bit in the off position.

Figure 1-11

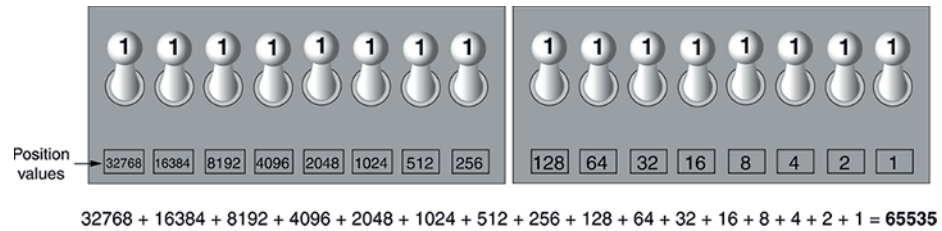
The bit pattern for 157



When all of the bits in a byte are set to 0 (turned off), then the value of the byte is 0. When all of the bits in a byte are set to 1 (turned on), then the byte holds the largest value that can be stored in it. The largest value that can be stored in a byte is $1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255$. This limit exists because there are only eight bits in a byte.

What if you need to store a number larger than 255? The answer is simple: use more than one byte. For example, suppose we put two bytes together. That gives us 16 bits. The position values of those 16 bits would be $2^0, 2^1, 2^2, 2^3$, and so forth, up through 2^{15} . As shown in **Figure 1-12**, the maximum value that can be stored in two bytes is 65,535. If you need to store a number larger than this, then more bytes are necessary.

Figure 1-12
Two bytes used for a large number



TIP: In case you're feeling overwhelmed by all this, relax! You will not have to actually convert numbers to binary while programming. Knowing that this process is

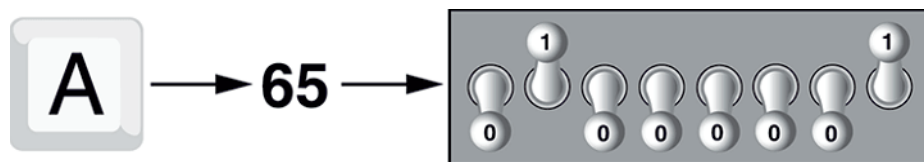
taking place inside the computer will help you as you learn, and in the long term this knowledge will make you a better programmer.

Storing Characters

Any piece of data that is stored in a computer's memory must be stored as a binary number. That includes characters, such as letters and punctuation marks. When a character is stored in memory, it is first converted to a numeric code. The numeric code is then stored in memory as a binary number.

Over the years, different coding schemes have been developed to represent characters in computer memory. Historically, the most important of these coding schemes is *ASCII*, which stands for the *American Standard Code for Information Interchange*. ASCII is a set of 128 numeric codes that represent the English letters, various punctuation marks, and other characters. For example, the ASCII code for the uppercase letter A is 65. When you type an uppercase A on your computer keyboard, the number 65 is stored in memory (as a binary number, of course). This is shown in **Figure 1-13**.

Figure 1-13
The letter A is stored in memory as the number 65



TIP: The acronym ASCII is pronounced "askee."

In case you are curious, the ASCII code for uppercase B is 66, for uppercase C is 67, and so forth. **Appendix C** shows all of the ASCII codes and the characters they represent.

The ASCII character set was developed in the early 1960s and was eventually adopted by almost all computer manufacturers. ASCII is limited, however, because it defines codes for only 128 characters. To remedy this, the Unicode character set was developed in the early 1990s. *Unicode* is an extensive encoding scheme that is compatible with ASCII, but can also represent characters for many of the languages in the world. Today, Unicode is quickly becoming the standard character set used in the computer industry.

Advanced Number Storage

Earlier, you read about numbers and how they are stored in memory. While reading that section, perhaps it occurred to you that the binary numbering system can be used to represent only integer numbers, beginning with 0. Negative numbers and real numbers (such as 3.14159) cannot be represented using the simple binary numbering technique we discussed.

Computers are able to store negative numbers and real numbers in memory, but to do so they use encoding schemes along with the binary numbering system. Negative numbers are encoded using a technique known as *two's complement*, and real numbers are encoded in *floating-point notation*. You don't need to know how these encoding schemes work, only that they are used to convert negative numbers and real numbers to binary format.

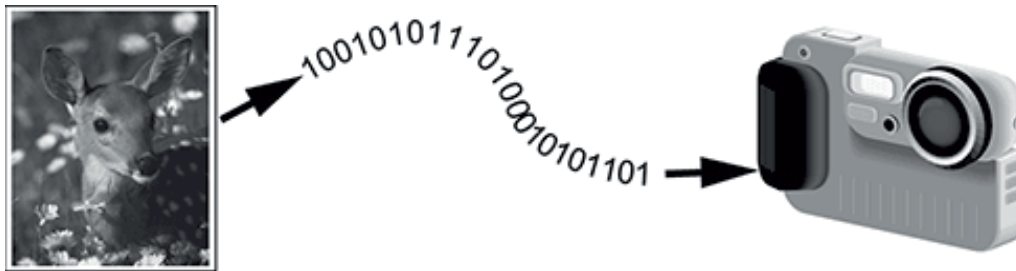
Other Types of Data

Computers are often referred to as digital devices. The term *digital* can be used to describe anything that uses binary numbers. *Digital data* is data that is stored in binary format, and a *digital device* is any device that works with binary data. In this section, we have discussed how numbers and characters are stored in binary, but computers also work with many other types of digital data.

For example, consider the pictures that you take with your digital camera. These images are composed of tiny dots of color known as *pixels*. (The term pixel stands for *picture element*.) As shown in **Figure 1-14**, each pixel in an image is converted to a numeric code that represents the pixel's color. The numeric code is stored in memory as a binary number.

Figure 1-14

A digital image is stored in binary format



Jupiterimages/PHOTOS.com/Getty Images

The music that you stream from an online source, or play on an MP3 player is also digital. A digital song is broken into small pieces known as *samples*. Each sample is converted to a binary number, which can be stored in memory. The more samples that a song is divided into, the more it sounds like the original music when it is played back. For example, a CD quality song is divided into more than 44,000 samples per second!

1.3: (Noninteractive) Checkpoint Questions from the Book

- 1.12 What amount of memory is enough to store a letter of the alphabet or a small number?
- 1.13 What do you call a tiny “switch” that can be set to either on or off?
- 1.14 In what numbering system are all numeric values written as sequences of 0s and 1s?
- 1.15 What is the purpose of ASCII?
- 1.16 What encoding scheme is extensive enough to represent the characters of many of the languages in the world?
- 1.17 What do the terms “digital data” and “digital device” mean?

1.4: How a Program Works

CONCEPT: A computer’s CPU can only understand instructions that are written in machine language. Because people find it very difficult to write entire programs in machine language, other programming languages have been invented.

Earlier, we stated that the CPU is the most important component in a computer because it is the part of the computer that runs programs. Sometimes the CPU is called the “computer’s brain” and is described as being “smart.” Although these are common metaphors, you should understand that the CPU is not a brain, and it is not smart. The CPU is an electronic device that is designed to do specific things. In particular, the CPU is designed to perform operations such as the following:

- Reading a piece of data from main memory
- Adding two numbers
- Subtracting one number from another number
- Multiplying two numbers
- Dividing one number by another number
- Moving a piece of data from one memory location to another
- Determining whether one value is equal to another value

As you can see from this list, the CPU performs simple operations on pieces of data. The CPU does nothing on its own, however. It has to be told what to do, and that’s the purpose of a program. A program is nothing more than a list of instructions that cause the CPU to perform operations.

Each instruction in a program is a command that tells the CPU to perform a specific operation. Here’s an example of an instruction that might appear in a program:

```
10110000
```

To you and me, this is only a series of 0s and 1s. To a CPU, however, this is an instruction to perform an operation.¹ It is written in 0s and 1s because CPUs only understand instructions that are written in *machine language*, and machine language instructions always have an underlying binary structure.

A machine language instruction exists for each operation that a CPU is capable of performing. For example, there is an instruction for adding numbers, there is an instruction for subtracting one number from another, and so forth. The entire set of instructions that a CPU can execute is known as the CPU's *instruction set*.

NOTE: There are several microprocessor companies today that manufacture CPUs. Some of the more well-known microprocessor companies are Intel, AMD, and Motorola. If you look carefully at your computer, you might find a tag showing a logo for its microprocessor.

Each brand of microprocessor has its own unique instruction set, which is typically understood only by microprocessors of the same brand. For example, Intel microprocessors understand the same instructions, but they do not understand instructions for Motorola microprocessors.

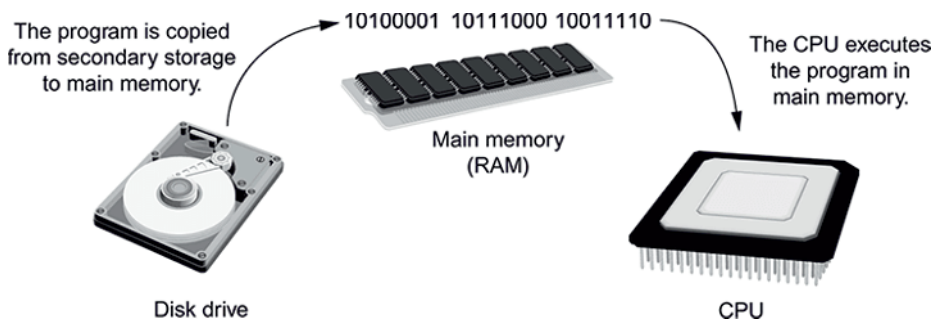
The machine language instruction that was previously shown is an example of only one instruction. It takes a lot more than one instruction, however, for the computer to do anything meaningful. Because the operations that a CPU knows how to perform are so basic in nature, a meaningful task can be accomplished only if the CPU performs many operations. For example, if you want your computer to calculate the amount of interest that you will earn from your savings account this year, the CPU will have to perform a large number of instructions, carried out in the proper sequence. It is not unusual for a program to contain thousands or even millions of machine language instructions.

Programs are usually stored on a secondary storage device such as a disk drive. When you install a program on your computer, the program is typically downloaded from a website, or installed from an online app store.

Although a program can be stored on a secondary storage device such as a disk drive, it has to be copied into main memory, or RAM, each time the CPU executes it. For example, suppose you have a word processing program on your computer's disk. To execute the program, you use the mouse to double-click the program's icon. This causes the program to be copied from the disk into main memory. Then, the computer's CPU executes the copy of the program that is in main memory. This process is illustrated in **Figure 1-15**.

Figure 1-15

A program is copied into main memory and then executed

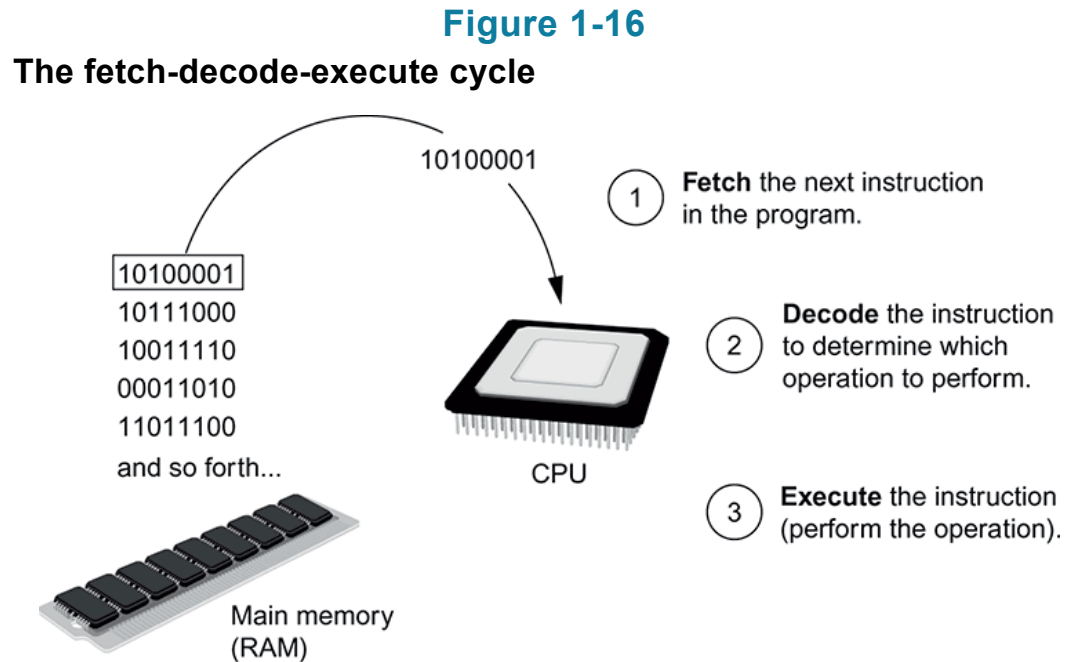


When a CPU executes the instructions in a program, it is engaged in a process that is known as the *fetch-decode-execute cycle*. This cycle, which consists of three steps, is repeated for each instruction in the program. The steps are:

1. **Fetch.** A program is a long sequence of machine language instructions. The first step of the cycle is to fetch, or read, the next instruction from memory into the CPU.

2. **Decode.** A machine language instruction is a binary number that represents a command that tells the CPU to perform an operation. In this step, the CPU decodes the instruction that was just fetched from memory, to determine which operation it should perform.
3. **Execute.** The last step in the cycle is to execute, or perform, the operation.

Figure 1-16 illustrates these steps.



From Machine Language to Assembly Language

Computers can only execute programs that are written in machine language. As previously mentioned, a program can have thousands or even millions of binary instructions, and writing such a program would be very tedious and time consuming. Programming in machine language would also be very difficult, because putting a 0 or a 1 in the wrong place will cause an error.

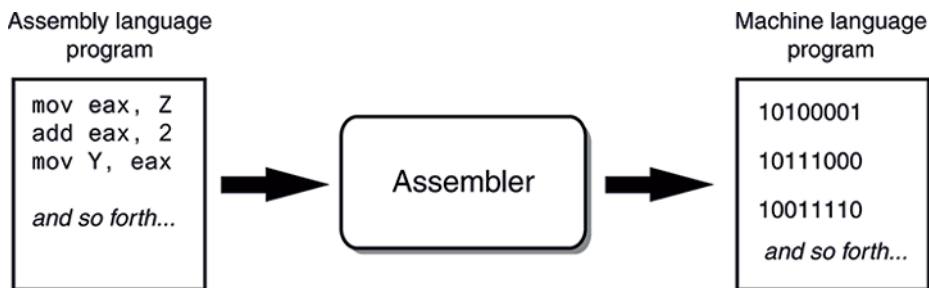
Although a computer's CPU only understands machine language, it is impractical for people to write programs in machine language. For this reason, *assembly language* was created in the early days of computing² as an alternative to machine language. Instead of using binary numbers for instructions, assembly language uses short words that are known as *mnemonics*. For example, in assembly language, the mnemonic `add` typically means to add numbers, `mul` typically means to multiply numbers, and `mov` typically means to move a value to a location in memory. When a programmer uses assembly language to write a program, he or she can write short mnemonics instead of binary numbers.

NOTE: There are many different versions of assembly language. It was mentioned earlier that each brand of CPU has its own machine language instruction set. Each brand of CPU typically has its own assembly language as well.

Assembly language programs cannot be executed by the CPU, however. The CPU only understands machine language, so a special program known as an *assembler* is used to translate an assembly language program to a machine language program. This process is shown in **Figure 1-17**. The machine language program that is created by the assembler can then be executed by the CPU.

Figure 1-17

An assembler translates an assembly language program to a machine language program



High-Level Languages

Although assembly language makes it unnecessary to write binary machine language instructions, it is not without difficulties. Assembly language is primarily a direct substitute for machine language, and like machine language, it requires that you know a lot about the CPU. Assembly language also requires that you write a large number of instructions for even the simplest program. Because assembly language is so close in nature to machine language, it is referred to as a *low-level language*.

In the 1950s, a new generation of programming languages known as *high-level languages* began to appear. A high-level language allows you to create powerful and complex programs without knowing how the CPU works and without writing large numbers of low-level instructions. In addition, most high-level languages use words that are easy to understand. For example, if a programmer were using COBOL (which was one of the early high-level languages created in the 1950s), he or she would write the following instruction to display the message *Hello world* on the computer screen:

```
DISPLAY "Hello world"
```

Python is a modern, high-level programming language that we will use in this book. In Python you would display the message *Hello world* with the following instruction:

```
print('Hello world')
```

Doing the same thing in assembly language would require several instructions and an intimate knowledge of how the CPU interacts with the computer's output device. As you can see from this example, high-level languages allow programmers to concentrate on the tasks they want to perform with their programs, rather than the details of how the CPU will execute those programs.

Since the 1950s, thousands of high-level languages have been created. **Table 1-1** lists several of the more well-known languages.

Table 1-1**Programming languages**

Language	Description
Ada	Ada was created in the 1970s, primarily for applications used by the U.S. Department of Defense. The language is named in honor of Ada Lovelace, a 19th century mathematician who published an algorithm that is considered by many to be the first computer program.
BASIC	B eginners A ll-purpose S ymbolic I nstruction C ode is a general-purpose language that was originally designed in the early 1960s to be simple enough for beginners to learn. Today, there are many different versions of BASIC.
FORTRAN	FOR mula TRAN slator was the first high-level programming language. It was designed in the 1950s for performing complex mathematical calculations.
COBOL	C ommon B usiness- O riented L anguage was created in the 1950s and was designed for business applications.
Pascal	Pascal was created in 1970 and was originally designed for teaching programming. The language was named in honor of the mathematician, physicist, and philosopher Blaise Pascal.
C and C++	C and C++ (pronounced “c plus plus”) are powerful, general-purpose languages developed at Bell Laboratories. The C language was created in 1972, and the C++ language was created in 1983.
C#	Pronounced “c sharp.” This language was created by Microsoft around the year 2000 for developing applications based on the Microsoft .NET platform.
Java	Java was created by Sun Microsystems in the early 1990s. It can be used to develop programs that run on a single computer or over the Internet from a web server.
JavaScript	JavaScript, created in the 1990s, can be used in Web pages. Despite its name, JavaScript is not related to Java.
Python	Python, the language we use in this book, is a general-purpose language created in the early 1990s. It has become popular in business and academic applications.

Ruby	Ruby is a general-purpose language that was created in the 1990s. It is increasingly becoming a popular language for programs that run on Web servers.
Rust	The Rust programming language is designed for high performance, memory safety, and concurrent execution. It was announced in 2010 by Mozilla Research.
Visual Basic	Visual Basic (commonly known as VB) is a Microsoft programming language and software development environment that allows programmers to create Windows-based applications quickly. VB was originally created in the early 1990s.

Keywords, Operators, and Syntax: An Overview

Each high-level language has its own set of predefined words that the programmer must use to write a program. The words that make up a high-level programming language are known as *keywords* or *reserved words*. Each keyword has a specific meaning, and cannot be used for any other purpose. **Table 1-2** shows all of the Python keywords.

Table 1-2

The Python keywords

and	continue	finally	is	raise
as	def	for	lambda	return
assert	del	from	None	True
async	elif	global	nonlocal	try
await	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield

In addition to keywords, programming languages have *operators* that perform various operations on data. For example, all programming languages have math operators that perform arithmetic. In Python, as well as most other languages, the + sign is an operator that adds two numbers. The following adds 12 and 75:

There are numerous other operators in the Python language, many of which you will learn about as you progress through this text.

In addition to keywords and operators, each language also has its own *syntax*, which is a set of rules that must be strictly followed when writing a program. The syntax rules dictate how keywords, operators, and various punctuation characters must be used in a program. When you are learning a programming language, you must learn the syntax rules for that particular language.

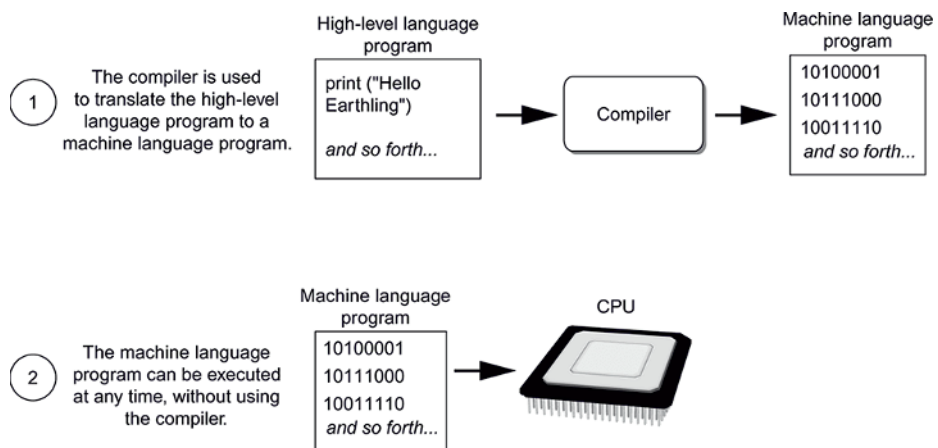
The individual instructions that you use to write a program in a high-level programming language are called *statements*. A programming statement can consist of keywords, operators, punctuation, and other allowable programming elements, arranged in the proper sequence to perform an operation.

Compilers and Interpreters

Because the CPU understands only machine language instructions, programs that are written in a high-level language must be translated into machine language. Depending on the language in which a program has been written, the programmer will use either a compiler or an interpreter to make the translation.

A *compiler* is a program that translates a high-level language program into a separate machine language program. The machine language program can then be executed any time it is needed. This is shown in **Figure 1-18**. As shown in the figure, compiling and executing are two different processes.

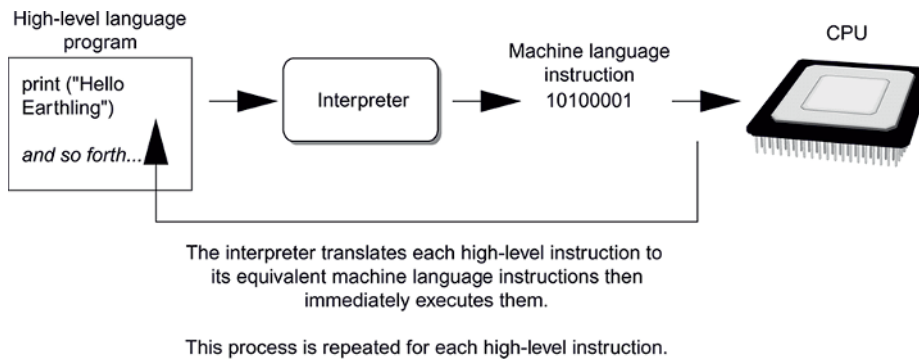
Figure 1-18
Compiling a high-level program and executing it



The Python language uses an *interpreter*, which is a program that both translates and executes the instructions in a high-level language program. As the interpreter reads each individual instruction in the program, it converts it to machine language instructions then immediately executes them. This process repeats for every instruction in the program. This process is illustrated in **Figure 1-19**. Because interpreters combine translation and execution, they typically do not create separate machine language programs.

Figure 1-19

Executing a high-level program with an interpreter



The statements that a programmer writes in a high-level language are called *source code*, or simply *code*. Typically, the programmer types a program’s code into a text editor then saves the code in a file on the computer’s disk. Next, the programmer uses a compiler to translate the code into a machine language program, or an interpreter to translate and execute the code. If the code contains a syntax error, however, it cannot be translated. A *syntax error* is a mistake such as a misspelled keyword, a missing punctuation character, or the incorrect use of an operator. When this happens, the compiler or interpreter displays an error message indicating that the program contains a syntax error. The programmer corrects the error then attempts once again to translate the program.

NOTE: Human languages also have syntax rules. Do you remember when you took your first English class, and you learned all those rules about commas, apostrophes, capitalization, and so forth? You were learning the syntax of the English language.

Although people commonly violate the syntax rules of their native language when speaking and writing, other people usually understand what they mean. Unfortunately, compilers and interpreters do not have this ability. If even a single syntax error appears in a program, the program cannot be compiled or executed. When an interpreter encounters a syntax error, it stops executing the program.

1.4: (Noninteractive) Checkpoint Questions from the Book

- 1.18 A CPU understands instructions that are written only in what language?
- 1.19 A program has to be copied into what type of memory each time the CPU executes it?
- 1.20 When a CPU executes the instructions in a program, it is engaged in what process?
- 1.21 What is assembly language?
- 1.22 What type of programming language allows you to create powerful and complex programs without knowing how the CPU works?
- 1.23 Each language has a set of rules that must be strictly followed when writing a program. What is this set of rules called?
- 1.24 What do you call a program that translates a high-level language program into a separate machine language program?
- 1.25 What do you call a program that both translates and executes the instructions in a high-level language program?
- 1.26 What type of mistake is usually caused by a misspelled keyword, a missing punctuation character, or the incorrect use of an operator?

1.5: Using Python

CONCEPT: The Python interpreter can run Python programs that are saved in files or interactively execute Python statements that are typed at the keyboard. Python comes with a program named IDLE that simplifies the process of writing, executing, and testing programs.

Installing Python

Before you can try any of the programs shown in this book, or write any programs of your own, you need to make sure that Python is installed on your computer and properly configured. If you are working in a computer lab, this has probably been done already. If you are using your own computer, you can follow the instructions in **Appendix A** to download and install Python.

The Python Interpreter

You learned earlier that Python is an interpreted language. When you install the Python language on your computer, one of the items that is installed is the Python interpreter. The *Python interpreter* is a program that can read Python programming statements and execute them. (Sometimes, we will refer to the Python interpreter simply as the interpreter.)

You can use the interpreter in two modes: interactive mode and script mode. In *interactive mode*, the interpreter waits for you to type Python statements on the keyboard. Once you type a statement, the interpreter executes it and then waits for you to type another statement. In *script mode*, the interpreter reads the contents of a file that contains Python statements. Such a file is known as a *Python program* or a *Python script*. The interpreter executes each statement in the Python program as it reads it.

Interactive Mode

Once Python has been installed and set up on your system, you start the interpreter in interactive mode by going to the operating system's command line and typing the following command:

```
python
```

If you are using Windows, you can alternatively type *python* in the Windows search box. In the search results, you will see a program named something like *Python 3.5*. (The "3.5" is the version of Python that is installed. At the time this is being written, Python 3.5 is the latest version.) Clicking this item will start the Python interpreter in interactive mode.

NOTE: When the Python interpreter is running in interactive mode, it is commonly called the *Python shell*.

When the Python interpreter starts in interactive mode, you will see something like the following displayed in a console window:

```
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May 3 2021, 17:27:52)
[MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

The `>>>` that you see is a prompt that indicates the interpreter is waiting for you to type a Python statement. Let's try it out. One of the simplest things that you can do in Python is print a message on the screen. For example, the following statement prints the message *Python programming is fun!* on the screen:

```
print('Python programming is fun!')
```

You can think of this as a command that you are sending to the Python interpreter. If you type the statement exactly as it is shown, the message *Python programming is fun!* is printed on the screen. Here is an example of how you type this statement at the interpreter's prompt:

```
>>> print('Python programming is fun!') [Enter]
```

After typing the statement, you press the Enter key, and the Python interpreter executes the statement, as shown here:

```
>>> print('Python programming is fun!') [Enter]
Python programming is fun!
>>>
```

After the message is displayed, the `>>>` prompt appears again, indicating the interpreter is waiting for you to enter another statement. Let's look at another example. In the following sample session, we have entered two statements:

```
>>> print('To be or not to be') [Enter]
To be or not to be
>>> print('That is the question.') [Enter]
That is the question.
>>>
```

If you incorrectly type a statement in interactive mode, the interpreter will display an error message. This will make interactive mode useful to you while you learn Python. As you learn new parts of the Python language, you can try them out in interactive mode and get immediate feedback from the interpreter.

To quit the Python interpreter in interactive mode on a Windows computer, press Ctrl-Z (pressing both keys together) followed by Enter. On a Mac, or Linux computer, press Ctrl-D.

NOTE: In **Chapter 2**, we will discuss the details of statements like the ones previously shown. If you want to try them now in interactive mode, make sure you type them exactly as shown.

- a. RowNumber
 - b. RowID
 - c. ID
 - d. RowIndex
27. A _____ key is two or more columns that are combined to create a unique value.
- a. combined
 - b. compound
 - c. composite
 - d. joined
28. The `sqlite3` module defines an exception named _____ that is thrown any time a database error occurs.
- a. `DBException`
 - b. `DataError`
 - c. `SQLException`
 - d. `Error`
29. This is a column in one table that references a primary key in another table.
- a. secondary key
 - b. fake key
 - c. foreign key
 - d. duplicate key

True or False

1. For storing large amounts of data, a DBMS is preferable to a traditional file.
2. All identity columns in a table must contain the same value.
3. A table can have more than one primary key.
4. When you connect to a database in SQLite, if the database does not exist an exception will be thrown.
5. When making changes to an SQLite database, those changes are not saved to the database until the `Connection` object's `commit` method is called.
6. With SQLite, you use the `Cursor` object's `execute` method to pass an SQL statement to the DBMS.
7. A database can have only one table.
8. It is possible to update more than one row with the SQL `UPDATE` statement.
9. When you create a table in SQLite, the `RowID` column is not automatically created.
10. In SQLite, when you designate a column as `INTEGER PRIMARY KEY`, that column becomes an alias for the `RowID` column.

Short Answer

1. If you are writing an application to store the customer and inventory records for a large business, why would you not want to use traditional text or binary files?

Index

&

&, 75, 881

*

* Multiplication, 15, 673

.

. Symbol, 907
.air, 243, 519, 527, 529, 541, 559, 892, 893
.au, 25
.mp3, 349
.mpg, 145

>

>, 403

A

ASCII, 14, 19, 37, 55, 163, 349, 861, 880, 888, 901
ASCII Character Set, 14, 19, 37, 861
Abstract, 573
Accessibility, 13
Accessors, 18, 621, 623
Accidents, 145
Accumulator, 227, 355, 371, 393, 395, 429, 431, 531, 886, 901
Accumulator Variable, 227, 429, 531
Accumulators, 17
Action, 12, 159, 167, 215, 325, 679, 895, 909
Adding, 15, 39, 81, 379, 429, 467, 519, 541, 573, 685, 689, 691, 729, 739, 769, 775, 831
Adding Elements To, 519
Address, 13, 15, 65, 571, 573, 605, 609, 615, 617, 619, 621, 629, 661, 765, 895, 907
Addresses, 571
Algebraic, 89
Algorithms, 18, 19, 658, 661
Alice, 863
Alignment, 169, 173, 882, 901
Animation, 123, 139, 189, 191, 193, 251
Apostrophe, 63
Appending Data To, 361
Applications, 16, 19, 43, 505, 621, 769, 771
Arithmetic, 43
Arrays, 407
Aspects, 19
Assembler, 41, 901
Assembly Language, 41, 45, 53, 880, 901, 911
Assessment, 767
Assignment Expression, 16, 185, 237, 901
Assignment Operator, 891, 901
Assignment Statements, 75, 143, 229
Association, 705, 888
Asterisk, 287
Asterisk *, 287
Authorization, 12
Automobiles, 635, 637
Average, 18, 85, 157, 231, 239, 257, 403, 429, 433, 435, 477, 479, 505, 515, 707, 709, 805

Background, 121, 139, 193, 884
Background Color, 121, 139, 193, 884
Bar Chart, 463, 465, 467, 471, 279

B

Base, 79, 161, 331, 635, 661, 663, 667, 669, 819, 845, 895, 901
Base Class, 901
Basis, 347
Binary Data, 37, 880
Binary Digit, 33, 901
Binary File, 349, 888, 901
Binary Files, 349, 845
Binary Numbering System, 37, 55, 880, 901
Binary Numbers, 37, 55, 880, 903
Bit, 33, 35, 51, 193, 661, 880, 901
Bits, 33
Block, 151, 155, 159, 169, 207, 217, 233, 247, 265, 267, 307, 389, 395, 579, 886, 887, 901, 903
Block Of Statements, 247, 265, 389, 395, 579, 903
Blocks, 169, 173
Blueprint, 577, 593, 894
Body, 211, 215, 237, 403, 885, 886, 901, 905
Books, 16, 25, 199
Bool, 181, 381, 383, 385
Boolean Condition, 185, 237
Boolean Expressions, 17, 153, 173, 175
Boolean Functions, 315
Boolean Values, 315
Boolean Variables, 149, 181, 499
Border, 685, 687, 689, 693
Borders, 685, 687, 689, 691
Braces, 517, 539, 553
Brackets, 405, 415, 439, 553
Branches, 393
Break, 17, 43, 91, 204, 245, 247, 260, 269, 323, 335
Break Statement, 245, 247
Browser, 347
Browsers, 33, 53
Buffer, 353
Bug, 15, 255, 911
Bugs, 255
Bus, 31
Businesses, 369, 769, 771, 897
Buttons, 19, 675, 699, 711, 713, 715, 717, 767, 896
Byte, 33, 35, 51, 880
Bytes, 33, 555, 893, 907, 909

C

COBOL, 41
CPU, 29, 31, 39, 41, 45, 51, 53, 880, 905, 911
Cables, 201
Callback Functions, 19
Calling A Function, 15, 263, 661
Calling Object, 583
Cancel, 129, 763
Cancel Button, 129
Capitalization, 45
Cards, 29, 297, 529, 531, 571
Case Sensitive, 881
Case Studies, 16, 22
Case-insensitive Comparisons, 163
Catalog, 577
Cell, 27, 597, 603
Cell Phone, 597, 603
Central Processing Unit, 29, 880
Certificate, 643, 645
Chapters, 20
Charts, 18, 271, 311, 451
Check, 19, 211, 631, 675, 711, 715, 717, 767, 829, 851, 853, 859, 896
Check Box, 715
Child, 197
Circles, 249, 251, 323
Circular, 31, 903
Class Attribute, 681
Class Definition, 579, 597, 627, 647, 655, 681
Class Method, 583, 627
Classes And, 14, 15, 18, 573, 593, 879, 911
Click, 39, 65, 129, 131, 697, 699,

721, 765, 767, 851, 852, 853, 855, 856, 857, 859
Clock, 237, 239, 575, 733, 888
Close A File, 353
Close Button, 852
Close Method, 353, 557, 898
Coding, 14, 22, 163, 387, 856
Command Line, 49, 131, 675, 677, 895
Commands, 109, 115, 121, 139, 675, 903
Comment, 65, 141, 265, 593, 903
Comments, 56, 65, 265, 856
Commit, 777, 779, 783, 789, 809, 813, 823, 831, 845, 898
Community, 11, 23, 24, 25, 55
Comparison, 213
Compiler, 45, 57, 880, 905
Components, 29, 33, 53, 880
Composite Key, 817, 899
Compression, 33
Computer Graphics, 575
Computer Security, 297
Computer Systems, 33
Computers, 13, 14, 17, 27, 29, 31, 33, 37, 163, 911
Concatenate, 93, 413, 489
Concatenation, 56, 93, 427, 485, 882
Conditional, 16, 149, 159, 183, 797, 811
Conditional Execution, 159
Conditional Expression, 183, 797, 811
Conditional Expressions, 16, 149
Conditions, 12, 167, 181, 193, 315
Constant, 109, 143, 157, 193, 293, 295, 309, 411, 603, 607, 719, 887
Constructors, 18
Contacts, 12, 607, 611, 773, 775, 777, 779
Content, 11, 12, 13, 24, 25, 797
Continuation, 91
Contract, 12
Control Structures, 16
Control Variable, 789
Conversion, 77, 91, 223, 401, 703, 767
Converting, 89, 555, 893
Cookie, 347, 577, 593, 625
Costs, 65
Count-controlled Loops, 17, 205
Counter Variable, 213, 909
Counters, 17
Currency, 73, 101, 876
Current Position, 123
Customers, 27, 199, 351, 511, 627, 783
Cycle, 17, 39, 56, 57, 658, 659, 880, 907

D

DBMS, 769, 771, 773, 775, 779, 781, 785, 793, 799, 815, 817, 829, 831, 833, 845, 897, 898
DROP, 779, 783, 898
DROP TABLE, 779, 783, 898
Data Hiding, 903
Data Processing, 337
Data Storage, 16
Data Structures, 18, 404
Data Type, 73, 75, 79, 143, 181, 711, 773, 817, 874, 903
Data Types, 17, 73, 181, 521, 573, 773
Databases, 19
Date, 503, 513, 515, 643, 645, 647
Dates, 515
Debugging, 173, 293, 887, 907
Decimal, 55, 81, 99, 101, 103, 105, 874, 875, 876, 877, 907
Decimal Numbers, 55
Decision Structures, 13, 14, 17, 149, 165, 167, 171, 177
Declarations, 907