GLOBAL EDITION

Intro to Python®

for Computer Science and Data Science

Paul Deitel • Harvey Deitel

Learning to Program with AI, Big Data and the Cloud

Digital Resources for Students

Your eBook provides 12-month access to digital resources that may include VideoNotes (step-by-step video tutorials on programming concepts), source code, and more. Refer to the preface in the textbook for a detailed list of resources.

Follow the instructions below to register for the Companion Website for Paul and Harvey Deitel's *Intro to Python®: for Computer Science and Data Science*, Global Edition.

1. Go to www.pearsonglobaleditions.com.
2. Enter the title of your textbook or browse by author name.
3. Click Companion Website.
4. Click Register and follow the on-screen instructions to create a login name and password.

ISSPCD-WAHOO-DIARY-KALPA-FRACK-OOSSE

Use the login name and password you created during registration to start using the online resources that accompany your textbook.

IMPORTANT:

This access code can only be used once. This subscription is valid for 12 months upon activation and is not transferable.

For technical support, go to https://support.pearson.com/getsupport.

Intro to Python® for Computer Science and Data Science

Learning to Program with AI, Big Data and the Cloud

by Paul Deitel & Harvey Deitel

PART 1
CS: Python Fundamentals Quickstart

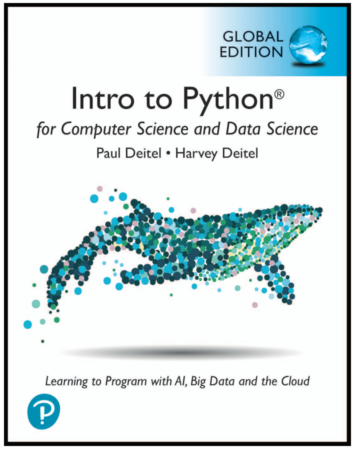
- **CS 1. Introduction to Computers and Python**
  - DS Intro: AI—at the Intersection of CS and DS
- **CS 2. Introduction to Python Programming**
  - DS Intro: Basic Descriptive Stats
- **CS 3. Control Statements and Program Development**
  - DS Intro: Measures of Central Tendency—Mean, Median, Mode
- **CS 4. Functions**
  - DS Intro: Basic Statistics—Measures of Dispersion
- **CS 5. Lists and Tuples**
  - DS Intro: Simulation and Static Visualization

PART 2
CS: Python Data Structures, Strings and Files

- **CS 6. Dictionaries and Sets**
  - DS Intro: Simulation and Dynamic Visualization
- **CS 7. Array-Oriented Programming with NumPy** High-Performance NumPy Arrays
  - DS Intro: Pandas Series and DataFrames
- **CS 8. Strings: A Deeper Look** Includes Regular Expressions
  - DS Intro: Pandas, Regular Expressions and Data Wrangling
- **CS 9. Files and Exceptions**
  - DS Intro: Loading Datasets from CSV Files into Pandas DataFrames

PART 3
CS: Python High-End Topics

- **CS 10. Object-Oriented Programming**
  - DS Intro: Time Series and Simple Linear Regression
- **CS 11. Computer Science Thinking: Recursion, Searching, Sorting and Big O**
- **CS and DS Other Topics Blog**



GLOBAL EDITION

Intro to Python®
for Computer Science and Data Science
Paul Deitel • Harvey Deitel

Learning to Program with AI, Big Data and the Cloud

PART 4
AI, Big Data and Cloud Case Studies

- **DS 12. Natural Language Processing (NLP)** Web Scraping in the Exercises
- **DS 13. Data Mining Twitter®** Sentiment Analysis, JSON and Web Services
- **DS 14. IBM Watson® and Cognitive Computing**
- **DS 15. Machine Learning: Classification, Regression and Clustering**
- **DS 16. Deep Learning** Convolutional and Recurrent Neural Networks; Reinforcement Learning in the Exercises
- **DS 17. Big Data: Hadoop®, Spark™, NoSQL and IoT**

1. Chapters 1–11 marked **CS** are traditional **Python** programming and **computer-science** topics.
2. Light-tinted bottom boxes in Chapters 1–10 marked **DS Intro** are **brief, friendly introductions to data-science topics**.
3. Chapters 12–17 marked **DS** are **Python-based**, **AI**, **big data** and **cloud** chapters, each containing **several full-implementation studies**.
4. **Functional-style programming** is integrated book wide.
5. Preface explains the dependencies among the chapters.
6. **Visualizations** throughout.
7. **CS courses** may cover more of the Python chapters and less of the DS content. Vice versa for **Data Science courses**.
8. We put Chapter 5 in Part 1. It's also a natural fit with Part 2.

Questions? `deitel@deitel.com`

Intro to Python®
for Computer Science and Data Science

Cover Designer: Straive

Cover Art: ©*Yuriy2012/Shutterstock*

Pearson Education Limited

KAO Two
KAO Park
Hockham Way
Harlow
CM17 9SR
United Kingdom

and Associated Companies throughout the world

Visit us on the World Wide Web at: www.pearsonglobaleditions.com

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

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

ISBN 10: 1-292-36490-4 (print)
ISBN 13: 978-1-292-36490-2 (print)
ISBN 13: 978-1-292-36493-3 (uPDF eBook)

GLOBAL EDITION

Intro to Python®

for Computer Science and Data Science

Paul Deitel • Harvey Deitel

Learning to Program with AI, Big Data and the Cloud

In Memory of Marvin Minsky,
a founding father of
artificial intelligence

It was a privilege to be your student in two
artificial-intelligence graduate courses at M.I.T.
You inspired your students to think beyond limits.

Harvey Deitel

Contents

Preface 19

Before You Begin 45

1 Introduction to Computers and Python 49

1.1 Introduction 50
1.2 Hardware and Software 51
 1.2.1 Moore's Law 52
 1.2.2 Computer Organization 52
1.3 Data Hierarchy 54
1.4 Machine Languages, Assembly Languages and High-Level Languages 57
1.5 Introduction to Object Technology 58
1.6 Operating Systems 61
1.7 Python 64
1.8 It's the Libraries! 66
 1.8.1 Python Standard Library 66
 1.8.2 Data-Science Libraries 66
1.9 Other Popular Programming Languages 68
1.10 Test-Drives: Using IPython and Jupyter Notebooks 69
 1.10.1 Using IPython Interactive Mode as a Calculator 69
 1.10.2 Executing a Python Program Using the IPython Interpreter 71
 1.10.3 Writing and Executing Code in a Jupyter Notebook 72
1.11 Internet and World Wide Web 77
 1.11.1 Internet: A Network of Networks 77
 1.11.2 World Wide Web: Making the Internet User-Friendly 78
 1.11.3 The Cloud 78
 1.11.4 Internet of Things 79
1.12 Software Technologies 80
1.13 How Big Is Big Data? 81
 1.13.1 Big Data Analytics 86
 1.13.2 Data Science and Big Data Are Making a Difference: Use Cases 87
1.14 Case Study—A Big-Data Mobile Application 88
1.15 Intro to Data Science: Artificial Intelligence—at the Intersection of CS and Data Science 90

2 Introduction to Python Programming 97

2.1 Introduction 98
2.2 Variables and Assignment Statements 98
2.3 Arithmetic 100
2.4 Function `print` and an Intro to Single- and Double-Quoted Strings 104
2.5 Triple-Quoted Strings 106
2.6 Getting Input from the User 107
2.7 Decision Making: The `if` Statement and Comparison Operators 109
2.8 Objects and Dynamic Typing 114
2.9 Intro to Data Science: Basic Descriptive Statistics 116
2.10 Wrap-Up 118

3 Control Statements and Program Development 121

3.1 Introduction 122
3.2 Algorithms 122
3.3 Pseudocode 123
3.4 Control Statements 123
3.5 `if` Statement 126
3.6 `if…else` and `if…elif…else` Statements 128
3.7 `while` Statement 133
3.8 `for` Statement 134
 3.8.1 Iterables, Lists and Iterators 136
 3.8.2 Built-In `range` Function 136
3.9 Augmented Assignments 137
3.10 Program Development: Sequence-Controlled Repetition 138
 3.10.1 Requirements Statement 138
 3.10.2 Pseudocode for the Algorithm 138
 3.10.3 Coding the Algorithm in Python 139
 3.10.4 Introduction to Formatted Strings 140
3.11 Program Development: Sentinel-Controlled Repetition 141
3.12 Program Development: Nested Control Statements 145
3.13 Built-In Function `range`: A Deeper Look 149
3.14 Using Type `Decimal` for Monetary Amounts 150
3.15 `break` and `continue` Statements 153
3.16 Boolean Operators `and`, `or` and `not` 154
3.17 Intro to Data Science: Measures of Central Tendency—
Mean, Median and Mode 157
3.18 Wrap-Up 159

4 Functions 167

4.1 Introduction 168
4.2 Defining Functions 168
4.3 Functions with Multiple Parameters 171
4.4 Random-Number Generation 173

4.5 Case Study: A Game of Chance 176
4.6 Python Standard Library 179
4.7 `math` Module Functions 180
4.8 Using IPython Tab Completion for Discovery 181
4.9 Default Parameter Values 183
4.10 Keyword Arguments 184
4.11 Arbitrary Argument Lists 184
4.12 Methods: Functions That Belong to Objects 186
4.13 Scope Rules 186
4.14 `import`: A Deeper Look 188
4.15 Passing Arguments to Functions: A Deeper Look 190
4.16 Function-Call Stack 193
4.17 Functional-Style Programming 194
4.18 Intro to Data Science: Measures of Dispersion 196
4.19 Wrap-Up 198

5 Sequences: Lists and Tuples 203

5.1 Introduction 204
5.2 Lists 204
5.3 Tuples 209
5.4 Unpacking Sequences 211
5.5 Sequence Slicing 214
5.6 `del` Statement 217
5.7 Passing Lists to Functions 219
5.8 Sorting Lists 220
5.9 Searching Sequences 222
5.10 Other List Methods 224
5.11 Simulating Stacks with Lists 226
5.12 List Comprehensions 227
5.13 Generator Expressions 229
5.14 Filter, Map and Reduce 230
5.15 Other Sequence Processing Functions 233
5.16 Two-Dimensional Lists 235
5.17 Intro to Data Science: Simulation and Static Visualizations 239
5.17.1 Sample Graphs for 600, 60,000 and 6,000,000 Die Rolls 239
5.17.2 Visualizing Die-Roll Frequencies and Percentages 241
5.18 Wrap-Up 247

6 Dictionaries and Sets 257

6.1 Introduction 258
6.2 Dictionaries 258
6.2.1 Creating a Dictionary 258
6.2.2 Iterating through a Dictionary 260
6.2.3 Basic Dictionary Operations 260

6.2.4 Dictionary Methods keys and values 262
6.2.5 Dictionary Comparisons 264
6.2.6 Example: Dictionary of Student Grades 265
6.2.7 Example: Word Counts 266
6.2.8 Dictionary Method update 268
6.2.9 Dictionary Comprehensions 268
6.3 Sets 269
6.3.1 Comparing Sets 271
6.3.2 Mathematical Set Operations 273
6.3.3 Mutable Set Operators and Methods 274
6.3.4 Set Comprehensions 276
6.4 Intro to Data Science: Dynamic Visualizations 276
6.4.1 How Dynamic Visualization Works 276
6.4.2 Implementing a Dynamic Visualization 279
6.5 Wrap-Up 282

7 Array-Oriented Programming with NumPy 287

7.1 Introduction 288
7.2 Creating arrays from Existing Data 289
7.3 array Attributes 290
7.4 Filling arrays with Specific Values 292
7.5 Creating arrays from Ranges 292
7.6 List vs. array Performance: Introducing %timeit 294
7.7 array Operators 296
7.8 NumPy Calculation Methods 298
7.9 Universal Functions 300
7.10 Indexing and Slicing 302
7.11 Views: Shallow Copies 304
7.12 Deep Copies 306
7.13 Reshaping and Transposing 307
7.14 Intro to Data Science: pandas Series and DataFrames 310
7.14.1 pandas Series 310
7.14.2 DataFrames 315
7.15 Wrap-Up 323

8 Strings: A Deeper Look 331

8.1 Introduction 332
8.2 Formatting Strings 333
8.2.1 Presentation Types 333
8.2.2 Field Widths and Alignment 334
8.2.3 Numeric Formatting 335
8.2.4 String's format Method 336
8.3 Concatenating and Repeating Strings 337
8.4 Stripping Whitespace from Strings 338

8.5 Changing Character Case 339
8.6 Comparison Operators for Strings 340
8.7 Searching for Substrings 340
8.8 Replacing Substrings 342
8.9 Splitting and Joining Strings 342
8.10 Characters and Character-Testing Methods 345
8.11 Raw Strings 346
8.12 Introduction to Regular Expressions 347
8.12.1 `re` Module and Function `fullmatch` 348
8.12.2 Replacing Substrings and Splitting Strings 351
8.12.3 Other Search Functions; Accessing Matches 352
8.13 Intro to Data Science: Pandas, Regular Expressions and Data Munging 355
8.14 Wrap-Up 360

9 Files and Exceptions 367

9.1 Introduction 368
9.2 Files 369
9.3 Text-File Processing 369
9.3.1 Writing to a Text File: Introducing the `with` Statement 370
9.3.2 Reading Data from a Text File 371
9.4 Updating Text Files 373
9.5 Serialization with JSON 375
9.6 Focus on Security: `pickle` Serialization and Deserialization 378
9.7 Additional Notes Regarding Files 378
9.8 Handling Exceptions 379
9.8.1 Division by Zero and Invalid Input 380
9.8.2 `try` Statements 380
9.8.3 Catching Multiple Exceptions in One `except` Clause 383
9.8.4 What Exceptions Does a Function or Method Raise? 384
9.8.5 What Code Should Be Placed in a `try` Suite? 384
9.9 `finally` Clause 384
9.10 Explicitly Raising an Exception 387
9.11 (Optional) Stack Unwinding and Tracebacks 387
9.12 Intro to Data Science: Working with CSV Files 390
9.12.1 Python Standard Library Module `csv` 390
9.12.2 Reading CSV Files into Pandas `DataFrames` 392
9.12.3 Reading the Titanic Disaster Dataset 394
9.12.4 Simple Data Analysis with the Titanic Disaster Dataset 395
9.12.5 Passenger Age Histogram 396
9.13 Wrap-Up 397

10 Object-Oriented Programming 403

10.1 Introduction 404
10.2 Custom Class `Account` 406

10.2.1 Test-Driving Class `Account` 406
10.2.2 `Account` Class Definition 408
10.2.3 Composition: Object References as Members of Classes 409
10.3 Controlling Access to Attributes 411
10.4 Properties for Data Access 412
10.4.1 Test-Driving Class `Time` 412
10.4.2 Class `Time` Definition 414
10.4.3 Class `Time` Definition Design Notes 418
10.5 Simulating "Private" Attributes 419
10.6 Case Study: Card Shuffling and Dealing Simulation 421
10.6.1 Test-Driving Classes `Card` and `DeckOfCards` 421
10.6.2 Class `Card`—Introducing Class Attributes 423
10.6.3 Class `DeckOfCards` 425
10.6.4 Displaying Card Images with Matplotlib 426
10.7 Inheritance: Base Classes and Subclasses 430
10.8 Building an Inheritance Hierarchy; Introducing Polymorphism 432
10.8.1 Base Class `CommissionEmployee` 432
10.8.2 Subclass `SalariedCommissionEmployee` 435
10.8.3 Processing `CommissionEmployees` and `SalariedCommissionEmployees` Polymorphically 439
10.8.4 A Note About Object-Based and Object-Oriented Programming 439
10.9 Duck Typing and Polymorphism 440
10.10 Operator Overloading 441
10.10.1 Test-Driving Class `Complex` 442
10.10.2 Class `Complex` Definition 443
10.11 Exception Class Hierarchy and Custom Exceptions 445
10.12 Named Tuples 447
10.13 A Brief Intro to Python 3.7's New Data Classes 448
10.13.1 Creating a `Card` Data Class 449
10.13.2 Using the `Card` Data Class 451
10.13.3 Data Class Advantages over Named Tuples 453
10.13.4 Data Class Advantages over Traditional Classes 454
10.14 Unit Testing with Docstrings and `doctest` 454
10.15 Namespaces and Scopes 459
10.16 Intro to Data Science: Time Series and Simple Linear Regression 462
10.17 Wrap-Up 471

11 Computer Science Thinking: Recursion, Searching, Sorting and Big O 479

11.1 Introduction 480
11.2 Factorials 481
11.3 Recursive Factorial Example 481
11.4 Recursive Fibonacci Series Example 484
11.5 Recursion vs. Iteration 487
11.6 Searching and Sorting 488

11.7 Linear Search 488
11.8 Efficiency of Algorithms: Big O 490
11.9 Binary Search 492
11.9.1 Binary Search Implementation 493
11.9.2 Big O of the Binary Search 495
11.10 Sorting Algorithms 496
11.11 Selection Sort 496
11.11.1 Selection Sort Implementation 497
11.11.2 Utility Function `print_pass` 498
11.11.3 Big O of the Selection Sort 499
11.12 Insertion Sort 499
11.12.1 Insertion Sort Implementation 500
11.12.2 Big O of the Insertion Sort 501
11.13 Merge Sort 502
11.13.1 Merge Sort Implementation 502
11.13.2 Big O of the Merge Sort 507
11.14 Big O Summary for This Chapter's Searching and Sorting Algorithms 507
11.15 Visualizing Algorithms 508
11.15.1 Generator Functions 510
11.15.2 Implementing the Selection Sort Animation 511
11.16 Wrap-Up 516

12 Natural Language Processing (NLP) 525

12.1 Introduction 526
12.2 TextBlob 527
12.2.1 Create a TextBlob 529
12.2.2 Tokenizing Text into Sentences and Words 530
12.2.3 Parts-of-Speech Tagging 530
12.2.4 Extracting Noun Phrases 531
12.2.5 Sentiment Analysis with TextBlob's Default Sentiment Analyzer 532
12.2.6 Sentiment Analysis with the `NaiveBayesAnalyzer` 534
12.2.7 Language Detection and Translation 535
12.2.8 Inflection: Pluralization and Singularization 537
12.2.9 Spell Checking and Correction 537
12.2.10 Normalization: Stemming and Lemmatization 538
12.2.11 Word Frequencies 539
12.2.12 Getting Definitions, Synonyms and Antonyms from WordNet 540
12.2.13 Deleting Stop Words 542
12.2.14 n-grams 544
12.3 Visualizing Word Frequencies with Bar Charts and Word Clouds 545
12.3.1 Visualizing Word Frequencies with Pandas 545
12.3.2 Visualizing Word Frequencies with Word Clouds 548
12.4 Readability Assessment with Textatistic 551
12.5 Named Entity Recognition with spaCy 553
12.6 Similarity Detection with spaCy 555

12.7 Other NLP Libraries and Tools 557
12.8 Machine Learning and Deep Learning Natural Language Applications 557
12.9 Natural Language Datasets 558
12.10 Wrap-Up 558

13 Data Mining Twitter 563

13.1 Introduction 564
13.2 Overview of the Twitter APIs 566
13.3 Creating a Twitter Account 567
13.4 Getting Twitter Credentials—Creating an App 568
13.5 What's in a Tweet? 569
13.6 Tweepy 573
13.7 Authenticating with Twitter Via Tweepy 573
13.8 Getting Information About a Twitter Account 575
13.9 Introduction to Tweepy Cursors: Getting an Account's Followers and Friends 577
13.9.1 Determining an Account's Followers 577
13.9.2 Determining Whom an Account Follows 580
13.9.3 Getting a User's Recent Tweets 580
13.10 Searching Recent Tweets 582
13.11 Spotting Trends: Twitter Trends API 584
13.11.1 Places with Trending Topics 584
13.11.2 Getting a List of Trending Topics 585
13.11.3 Create a Word Cloud from Trending Topics 587
13.12 Cleaning/Preprocessing Tweets for Analysis 589
13.13 Twitter Streaming API 590
13.13.1 Creating a Subclass of `StreamListener` 591
13.13.2 Initiating Stream Processing 593
13.14 Tweet Sentiment Analysis 595
13.15 Geocoding and Mapping 599
13.15.1 Getting and Mapping the Tweets 600
13.15.2 Utility Functions in `tweetutilities.py` 604
13.15.3 Class `LocationListener` 606
13.16 Ways to Store Tweets 607
13.17 Twitter and Time Series 608
13.18 Wrap-Up 608

14 IBM Watson and Cognitive Computing 613

14.1 Introduction: IBM Watson and Cognitive Computing 614
14.2 IBM Cloud Account and Cloud Console 616
14.3 Watson Services 616
14.4 Additional Services and Tools 620
14.5 Watson Developer Cloud Python SDK 621
14.6 Case Study: Traveler's Companion Translation App 622

14.6.1 Before You Run the App 623
14.6.2 Test-Driving the App 624
14.6.3 `SimpleLanguageTranslator.py` Script Walkthrough 625
14.7 Watson Resources 635
14.8 Wrap-Up 637

15 Machine Learning: Classification, Regression and Clustering 641
15.1 Introduction to Machine Learning 642
15.1.1 Scikit-Learn 643
15.1.2 Types of Machine Learning 644
15.1.3 Datasets Bundled with Scikit-Learn 646
15.1.4 Steps in a Typical Data Science Study 647
15.2 Case Study: Classification with k-Nearest Neighbors and the Digits Dataset, Part 1 647
15.2.1 k-Nearest Neighbors Algorithm 649
15.2.2 Loading the Dataset 650
15.2.3 Visualizing the Data 654
15.2.4 Splitting the Data for Training and Testing 656
15.2.5 Creating the Model 657
15.2.6 Training the Model 658
15.2.7 Predicting Digit Classes 658
15.3 Case Study: Classification with k-Nearest Neighbors and the Digits Dataset, Part 2 660
15.3.1 Metrics for Model Accuracy 660
15.3.2 K-Fold Cross-Validation 664
15.3.3 Running Multiple Models to Find the Best One 665
15.3.4 Hyperparameter Tuning 667
15.4 Case Study: Time Series and Simple Linear Regression 668
15.5 Case Study: Multiple Linear Regression with the California Housing Dataset 673
15.5.1 Loading the Dataset 674
15.5.2 Exploring the Data with Pandas 676
15.5.3 Visualizing the Features 678
15.5.4 Splitting the Data for Training and Testing 682
15.5.5 Training the Model 682
15.5.6 Testing the Model 683
15.5.7 Visualizing the Expected vs. Predicted Prices 684
15.5.8 Regression Model Metrics 685
15.5.9 Choosing the Best Model 686
15.6 Case Study: Unsupervised Machine Learning, Part 1—Dimensionality Reduction 687
15.7 Case Study: Unsupervised Machine Learning, Part 2—k-Means Clustering 690
15.7.1 Loading the Iris Dataset 692

15.7.2 Exploring the Iris Dataset: Descriptive Statistics with Pandas 694
15.7.3 Visualizing the Dataset with a Seaborn `pairplot` 695
15.7.4 Using a `KMeans` Estimator 698
15.7.5 Dimensionality Reduction with Principal Component Analysis 700
15.7.6 Choosing the Best Clustering Estimator 703
15.8 Wrap-Up 704

16 Deep Learning 713
16.1 Introduction 714
16.1.1 Deep Learning Applications 716
16.1.2 Deep Learning Demos 717
16.1.3 Keras Resources 717
16.2 Keras Built-In Datasets 717
16.3 Custom Anaconda Environments 718
16.4 Neural Networks 720
16.5 Tensors 722
16.6 Convolutional Neural Networks for Vision; Multi-Classification with the MNIST Dataset 724
16.6.1 Loading the MNIST Dataset 725
16.6.2 Data Exploration 726
16.6.3 Data Preparation 728
16.6.4 Creating the Neural Network 730
16.6.5 Training and Evaluating the Model 739
16.6.6 Saving and Loading a Model 744
16.7 Visualizing Neural Network Training with TensorBoard 745
16.8 ConvnetJS: Browser-Based Deep-Learning Training and Visualization 748
16.9 Recurrent Neural Networks for Sequences; Sentiment Analysis with the IMDb Dataset 749
16.9.1 Loading the IMDb Movie Reviews Dataset 750
16.9.2 Data Exploration 751
16.9.3 Data Preparation 753
16.9.4 Creating the Neural Network 754
16.9.5 Training and Evaluating the Model 757
16.10 Tuning Deep Learning Models 758
16.11 Convnet Models Pretrained on ImageNet 759
16.12 Reinforcement Learning 760
16.12.1 Deep Q-Learning 761
16.12.2 OpenAI Gym 761
16.13 Wrap-Up 762

17 Big Data: Hadoop, Spark, NoSQL and IoT 771
17.1 Introduction 772
17.2 Relational Databases and Structured Query Language (SQL) 776
17.2.1 A `books` Database 778

17.2.2 SELECT Queries 782
17.2.3 WHERE Clause 782
17.2.4 ORDER BY Clause 784
17.2.5 Merging Data from Multiple Tables: INNER JOIN 785
17.2.6 INSERT INTO Statement 786
17.2.7 UPDATE Statement 787
17.2.8 DELETE FROM Statement 787
17.3 NoSQL and NewSQL Big-Data Databases: A Brief Tour 789
17.3.1 NoSQL Key–Value Databases 789
17.3.2 NoSQL Document Databases 790
17.3.3 NoSQL Columnar Databases 790
17.3.4 NoSQL Graph Databases 791
17.3.5 NewSQL Databases 791
17.4 Case Study: A MongoDB JSON Document Database 792
17.4.1 Creating the MongoDB Atlas Cluster 793
17.4.2 Streaming Tweets into MongoDB 794
17.5 Hadoop 803
17.5.1 Hadoop Overview 803
17.5.2 Summarizing Word Lengths in *Romeo and Juliet* via MapReduce 806
17.5.3 Creating an Apache Hadoop Cluster in Microsoft Azure HDInsight 806
17.5.4 Hadoop Streaming 808
17.5.5 Implementing the Mapper 808
17.5.6 Implementing the Reducer 809
17.5.7 Preparing to Run the MapReduce Example 810
17.5.8 Running the MapReduce Job 811
17.6 Spark 814
17.6.1 Spark Overview 814
17.6.2 Docker and the Jupyter Docker Stacks 815
17.6.3 Word Count with Spark 818
17.6.4 Spark Word Count on Microsoft Azure 821
17.7 Spark Streaming: Counting Twitter Hashtags Using the pyspark-notebook Docker Stack 825
17.7.1 Streaming Tweets to a Socket 825
17.7.2 Summarizing Tweet Hashtags; Introducing Spark SQL 828
17.8 Internet of Things and Dashboards 834
17.8.1 Publish and Subscribe 836
17.8.2 Visualizing a PubNub Sample Live Stream with a Freeboard Dashboard 836
17.8.3 Simulating an Internet-Connected Thermostat in Python 838
17.8.4 Creating the Dashboard with Freeboard.io 840
17.8.5 Creating a Python PubNub Subscriber 842
17.9 Wrap-Up 846

Index 853

This page is intentionally left blank

Preface

"There's gold in them thar hills!"[1]

For many decades, some powerful trends have been in place. Computer hardware has rapidly been getting faster, cheaper and smaller. Internet bandwidth (that is, its information carrying capacity) has rapidly been getting larger and cheaper. And quality computer software has become ever more abundant and essentially free or nearly free through the "open source" movement. Soon, the "Internet of Things" will connect tens of billions of devices of every imaginable type. These will generate enormous volumes of data at rapidly increasing speeds and quantities.

Not so many years ago, if people had told us that we'd write a college-level introductory programming textbook with words like "Big Data" and "Cloud" in the title and a graphic of a multicolored whale (emblematic of "big") on the cover, our reaction might have been, "Huh?" And, if they'd told us we'd include AI (for artificial intelligence) in the title, we might have said, "Really? Isn't that pretty advanced stuff for novice programmers?"

If people had said, we'd include "Data Science" in the title, we might have responded, "Isn't data already included in the domain of 'Computer Science'? Why would we need a separate academic discipline for it?" Well, in programming today, the latest innovations are "all about the data"—*data* science, *data* analytics, big *data*, relational *data*bases (SQL), and NoSQL and NewSQL *data*bases.

So, here we are! Welcome to *Intro to Python for Computer Science and Data Science: Learning to Program with AI, Big Data and the Cloud.*

In this book, you'll learn hands-on with today's most compelling, leading-edge computing technologies—and, as you'll see, with an easily tunable mix of computer science and data science appropriate for introductory courses in those and related disciplines. And, you'll program in Python—one of the world's most popular languages and the fastest growing among them. In this Preface, we present the "soul of the book."

Professional programmers often quickly discover that they like Python. They appreciate its expressive power, readability, conciseness and interactivity. They like the world of open-source software development that's generating an ever-growing base of reusable software for an enormous range of application areas.

Whether you're an instructor, a novice student or an experienced professional programmer, this book has much to offer you. Python is an excellent first programming language for novices and is equally appropriate for developing industrial-strength applications. For the novice, the early chapters establish a solid programming foundation.

We hope you'll find *Intro to Python for Computer Science and Data Science* educational, entertaining and challenging. It has been a joy to work on this project.

1. Source unknown, frequently misattributed to Mark Twain.

Python for Computer Science and Data Science Education

Many top U.S. universities have switched to Python as their language of choice for teaching introductory computer science, with "eight of the top 10 CS departments (80%), and 27 of the top 39 (69%)" using Python.[2] It's now particularly popular for educational and scientific computing,[3] and it recently surpassed R as the most popular data science programming language.[4,5,6]

Modular Architecture

We anticipate that the computer science undergraduate curriculum will evolve to include a data science component—this book is designed to facilitate that and to meet the needs of introductory data science courses with a Python programming component.

The book's **modular architecture** (please see the **Table of Contents graphic** on the book's first page) helps us meet the diverse needs of computer science, data science and related audiences. Instructors can adapt it conveniently to a wide range of courses offered to **students drawn from many majors**.

Chapters 1–11 cover traditional introductory computer science programming topics. Chapters 1–10 each include an *optional* brief **Intro to Data Science** section introducing artificial intelligence, basic descriptive statistics, measures of central tendency and dispersion, simulation, static and dynamic visualization, working with CSV files, pandas for data exploration and data wrangling, time series and simple linear regression. These help you prepare for the data science, AI, big data and cloud case studies in Chapters 12–17, which present opportunities for you to use **real-world datasets** in complete case studies.

After covering Python Chapters 1–5 and a few key parts of Chapters 6–7, you'll be able to handle significant portions of the **data science, AI and big data case studies** in Chapters 12–17, which are appropriate for all contemporary programming courses:

- Computer science courses will likely work through more of Chapters 1–11 and fewer of the Intro to Data Science sections in Chapters 1–10. CS instructors will want to cover some or all of the case-study Chapters 12–17.
- Data science courses will likely work through fewer of Chapters 1–11, most or all of the Intro to Data Science sections in Chapters 1–10, and most or all of the case-study Chapters 12–17.

The "Chapter Dependencies" section of this Preface will help instructors plan their syllabi in the context of the book's unique architecture.

Chapters 12–17 are loaded with cool, powerful, contemporary content. They present hands-on implementation case studies on topics such as supervised machine learning, unsupervised machine learning, deep learning, reinforcement learning (in the exercises), natural

2. Guo, Philip., "Python Is Now the Most Popular Introductory Teaching Language at Top U.S. Universities," ACM, July 07, 2014, `https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext`.
3. `https://www.oreilly.com/ideas/5-things-to-watch-in-python-in-2017`.
4. `https://www.kdnuggets.com/2017/08/python-overtakes-r-leader-analytics-data-science.html`.
5. `https://www.r-bloggers.com/data-science-job-report-2017-r-passes-sas-but-python-leaves-them-both-behind/`.
6. `https://www.oreilly.com/ideas/5-things-to-watch-in-python-in-2017`.

language processing, data mining Twitter, cognitive computing with IBM's Watson, big data and more. Along the way, you'll acquire a **broad literacy** of data science terms and concepts, ranging from briefly defining terms to using concepts in small, medium and large programs. Browsing the book's detailed index will give you a sense of the breadth of coverage.

Audiences for the Book

The modular architecture makes this book appropriate for several audiences:

- **All standard Python computer science and related majors**. First and foremost, our book is a solid contemporary Python CS 1 entry. The computing curriculum recommendations from the ACM/IEEE list five types of computing programs: Computer Engineering, Computer Science, Information Systems, Information Technology and Software Engineering.[7] The book is appropriate for each of these.
- **Undergraduate courses for data science majors**—Our book is useful in many data science courses. It follows the curriculum recommendations for **integration of all the key areas in all courses**, as appropriate for intro courses. In the proposed data science curriculum, the book can be the primary textbook for the first computer science course or the first data science course, then be used as a Python reference throughout the upper curriculum.
- **Service courses** for students who are not computer science or data science majors.
- **Graduate courses in data science**—The book can be used as the primary textbook in the first course, then as a Python reference in other graduate-level data science courses.
- **Two-year colleges**—These schools will increasingly offer courses that prepare students for data science programs in the four-year colleges—the book is an appropriate option for that purpose.
- **High schools**—Just as they began teaching computer classes in response to strong interest, many are already teaching Python programming and data science classes.[8] According to a recent article on LinkedIn, "data science should be taught in high school," where the "curriculum should mirror the types of careers that our children will go into, focused directly on where jobs and technology are going."[9] We believe that data science could soon become a popular college advanced-placement course and that eventually there will be a data science AP exam.
- **Professional industry training courses**.

Key Features

KIS (Keep It Simple), KIS (Keep it Small), KIT (Keep it Topical)

- **Keep it simple**—In every aspect of the book and its instructor and student supplements, we strive for **simplicity and clarity**. For example, when we present nat-

7. `https://www.acm.org/education/curricula-recommendations`.
8. `http://datascience.la/introduction-to-data-science-for-high-school-students/`.
9. `https://www.linkedin.com/pulse/data-science-should-taught-high-school-rebecca-croucher/`.

ural language processing, we use the simple and intuitive TextBlob library rather than the more complex NLTK. In general, when multiple libraries could be used to perform similar tasks, we use the simplest one.

- **Keep it small**—Most of the book's 538 examples are small—often just a few lines of code, with immediate interactive IPython feedback. We use large examples as appropriate in approximately 40 larger scripts and complete case studies.
- **Keep it topical**—We read scores of recent Python-programming and data science textbooks and professional books. In all we browsed, read or watched about 15,000 current articles, research papers, white papers, videos, blog posts, forum posts and documentation pieces. This enabled us to "take the pulse" of the Python, computer science, data science, AI, big data and cloud communities to create 1566 up-to-the-minute examples, exercises and projects (EEPs).

IPython's Immediate-Feedback, Explore, Discover and Experiment Pedagogy

- The ideal way to learn from this book is to read it and run the code examples in parallel. Throughout the book, we use the **IPython interpreter**, which provides a friendly, immediate-feedback, interactive mode for quickly exploring, discovering and experimenting with Python and its extensive libraries.
- Most of the code is presented in **small, interactive IPython sessions** (which we call **IIs**). For each code snippet you write, IPython immediately reads it, evaluates it and prints the results. This **instant feedback** keeps your attention, boosts learning, facilitates rapid prototyping and speeds the software-development process.
- Our books always emphasize the **live-code teaching approach**, focusing on *complete, working programs* with *sample inputs and outputs*. IPython's "magic" is that it turns snippets into live code that "comes alive" as you enter each line. This promotes learning and encourages experimentation.
- IPython is a great way to learn the error messages associated with common errors. We'll intentionally make errors to show you what happens. When we say something is an error, try it to see what happens.
- We use this same immediate-feedback philosophy in the book's 557 **Self-Check Exercises** (ideal for "flipped classrooms"—we'll soon say more about that phenomenon) and many of the 471 end-of-chapter exercises and projects.

Python Programming Fundamentals

- First and foremost, this is an introductory Python textbook. We provide rich coverage of Python and general programming fundamentals.
- We discuss Python's programming models—**procedural programming**, **functional-style programming** and **object-oriented programming**.
- We emphasize **problem-solving** and **algorithm development**.
- We use best practices to **prepare students for industry**.
- **Functional-style programming** is used throughout the book as appropriate. A chart in Chapter 4 lists most of Python's key functional-style programming capabilities and the chapters in which we initially cover many of them.

538 Examples, and 471 Exercises and Projects (EEPs)

- Students use a hands-on applied approach to learn from a broad selection of **real-world examples, exercises and projects** (EEPs) drawn from computer science, data science and many other fields.
- The **538 examples** range from individual code snippets to complete computer science, data science, artificial intelligence and big data case studies.
- The **471 exercises and projects** naturally extend the chapter examples. Each chapter concludes with a substantial set of exercises covering a wide variety of topics. This helps instructors tailor their courses to the unique requirements of their audiences and to vary course assignments each semester.
- The EEPs give you an engaging, challenging and entertaining introduction to Python programming, including hands-on AI, computer science and data science.
- Students attack exciting and entertaining challenges with **AI, big data and cloud** technologies like **natural language processing**, **data mining Twitter**, **machine learning, deep learning, Hadoop**, **MapReduce**, **Spark**, **IBM Watson**, key data science libraries (**NumPy**, **pandas**, **SciPy**, **NLTK**, **TextBlob**, **spaCy**, **BeautifulSoup**, **Textatistic**, **Tweepy**, **Scikit-learn**, **Keras**), key visualization libraries (**Matplotlib**, **Seaborn**, **Folium**) and more.
- Our EEPs encourage you to think into the future. We had the following idea as we wrote this Preface—although it's not in the text, many similar thought-provoking projects are: With **deep learning**, the **Internet of Things** and large numbers of TV cameras trained on sporting events, it will become possible to keep *automatic statistics*, review the details of every play and resolve instant-replay reviews immediately. So, fans won't have to endure the bad calls and delays common in today's sporting events. Here's a thought—we can use these technologies to eliminate referees. Why not? We're increasingly entrusting our lives to other deep-learning-based technologies like **robotic surgeons** and **self-driving cars**!
- The **project exercises** encourage you to go deeper into what you've learned and research technologies we have not covered. Projects are often larger in scope and may require significant Internet research and implementation effort.
- In the **instructor supplements**, we provide solutions to many exercises, including most in the core Python Chapters 1–11. **Solutions are available only to instructors**—see the section "Instructor Supplements on Pearson's Instructor Resource Center" later in this Preface for details. **We do not provide solutions to the project and research exercises.**
- We encourage you to look at lots of **demos** and free **open-source** code examples (available on sites such as **GitHub**) for inspiration on additional **class projects, term projects**, **directed-study projects**, **capstone-course projects** and **thesis research**.

557 Self-Check Exercises and Answers

- Most sections end with an average of three **Self-Check Exercises**.
- **Fill-in-the-blank, true/false and discussion Self Checks** enable you to test your understanding of the concepts you just studied.

- **IPython interactive Self Checks** give you a chance to try out and reinforce the programming techniques you just learned.
- For rapid learning, answers immediately follow all Self-Check Exercises.

Avoid Heavy Math in Favor of English Explanations

- Data science topics can be highly mathematical. This book will be used in first computer science and data science courses where students may not have deep mathematical backgrounds, so we avoid heavy math, leaving it to upper-level courses.
- We capture the conceptual essence of the mathematics and put it to work in our examples, exercises and projects. We do this by using **Python libraries** such as **statistics**, **NumPy**, **SciPy**, **pandas** and many others, which hide the mathematical complexity. So, it's straightforward for students to get many of the benefits of mathematical techniques like **linear regression** without having to know the mathematics behind them. In the **machine-learning** and **deep-learning** examples, we focus on creating objects that do the math for you "behind the scenes." This is one of the keys to **object-*based* programming**. It's like driving a car safely to your destination without knowing all the math, engineering and science that goes into building engines, transmissions, power steering and anti-skid braking systems.

Visualizations

- 67 **full-color static, dynamic, animated and interactive two-dimensional and three-dimensional visualizations** (charts, graphs, pictures, animations etc.) help you understand concepts.
- We focus on high-level visualizations produced by **Matplotlib**, **Seaborn**, **pandas** and **Folium** (for **interactive maps**).
- We use visualizations as a pedagogic tool. For example, we make the **law of large numbers** "come alive" in a dynamic **die-rolling simulation** and bar chart. As the number of rolls increases, you'll see each face's percentage of the total rolls gradually approach 16.667% (1/6th) and the sizes of the bars representing the percentages equalize.
- You need to get to know your data. One way is simply to look at the raw data. For even modest amounts of data, you could rapidly get lost in the detail. Visualizations are especially crucial in big data for **data exploration** and **communicating reproducible research results**, where the data items can number in the millions, billions or more. A common saying is that a picture is worth a thousand words[10]—in **big data**, a visualization could be worth billions or more items in a database.
- Sometimes, you need to "fly 40,000 feet above the data" to see it "in the large." **Descriptive statistics** help but can be misleading. Anscombe's quartet, which you'll investigate in the exercises, demonstrates through visualizations that significantly *different* datasets can have *nearly identical* descriptive statistics.
- We show the visualization and animation code so you can implement your own. We also provide the animations in source-code files and as Jupyter Notebooks, so

10. `https://en.wikipedia.org/wiki/A_picture_is_worth_a_thousand_words`.

you can conveniently customize the code and animation parameters, re-execute the animations and see the effects of the changes.

- Many exercises ask you to create your own visualizations.

Data Experiences

- The undergraduate data science curriculum proposal says "**Data experiences** need to play a central role in all courses."[11]
- In the book's examples, exercises and projects (EEPs), you'll work with many **real-world datasets and data sources**. There's a wide variety of **free open datasets** available online for you to experiment with. Some of the sites we reference list hundreds or thousands of datasets. We encourage you to explore these.
- We collected hundreds of syllabi, tracked down **instructor dataset preferences** and researched the most popular datasets for **supervised machine learning**, **unsupervised machine learning** and **deep learning** studies. Many of the libraries you'll use come bundled with popular datasets for experimentation.
- You'll learn the steps required to obtain data and prepare it for analysis, analyze that data using many techniques, tune your models and communicate your results effectively, especially through visualization.

Thinking Like a Developer

- You'll work with a **developer focus**, using such popular sites as **GitHub** and **StackOverflow**, and doing lots of Internet research. Our **Intro to Data Science sections** and case studies in Chapters 12–17 provide rich data experiences.
- **GitHub** is an excellent venue for **finding open-source code** to incorporate into your projects (and to contribute your code to the **open-source community**). It's also a crucial element of the software developer's arsenal with **version control tools** that help teams of developers manage open-source (and private) projects.
- We encourage you to study developers' code on sites like GitHub.
- To get ready for career work in computer science and data science, you'll use an extraordinary range of free and open-source Python and data science **libraries**, free and open **real-world datasets** from government, industry and academia, and **free**, **free-trial** and **freemium** offerings of software and cloud services.

Hands-On Cloud Computing

- Much of big data analytics occurs in the cloud, where it's easy to scale *dynamically* the amount of hardware and software your applications need. You'll work with various cloud-based services (some directly and some indirectly), including Twitter, Google Translate, IBM Watson, Microsoft Azure, OpenMapQuest, geopy, Dweet.io and PubNub. You'll explore more in the exercises and projects.
- We encourage you to use free, free trial or freemium services from various cloud vendors. We prefer those that don't require a credit card because you don't want

11. "Curriculum Guidelines for Undergraduate Programs in Data Science," `http://www.annualreviews.org/doi/full/10.1146/annurev-statistics-060116-053930` (p. 18).

to risk accidentally running up big bills. **If you decide to use a service that requires a credit card, ensure that the tier you're using for free will not automatically jump to a paid tier.**

Database, Big Data and Big Data Infrastructure

- According to IBM (Nov. 2016), 90% of the world's data was created in the last two years.[12] Evidence indicates that the speed of data creation is accelerating.
- According to a March 2016 *AnalyticsWeek* article, within five years there will be over 50 billion devices connected to the Internet and by 2020 we'll be producing 1.7 megabytes of new data every second *for every person on the planet*![13]
- We include an optional treatment of **relational databases** and **SQL** with **SQLite**.
- Databases are critical big data infrastructure for storing and manipulating the massive amounts of data you'll process. Relational databases process *structured data*—they're not geared to the *unstructured* and *semi-structured data* in big data applications. So, as big data evolved, NoSQL and NewSQL databases were created to handle such data efficiently. We include a **NoSQL** and **NewSQL** overview and a hands-on case study with a **MongoDB JSON document database**.
- We include a solid treatment of **big data hardware and software infrastructure** in Chapter 17, "Big Data: Hadoop, Spark, NoSQL and IoT (Internet of Things)."

Artificial Intelligence Case Studies

- Why doesn't this book have an artificial intelligence chapter? After all, AI is on the cover. In the case study Chapters 12–16, we present **artificial intelligence** topics (a key intersection between computer science and data science), including **natural language processing**, **data mining Twitter to perform sentiment analysis**, **cognitive computing with IBM Watson**, **supervised machine learning**, **unsupervised machine learning**, **deep learning** and **reinforcement learning** (in the exercises). Chapter 17 presents the big data hardware and software infrastructure that enables computer scientists and data scientists to implement leading-edge AI-based solutions.

Computer Science

- The Python fundamentals treatment in Chapters 1–10 will get you thinking like a computer scientist. Chapter 11, "Computer Science Thinking: Recursion, Searching, Sorting and Big O," gives you a more advanced perspective—these are classic computer science topics. Chapter 11 emphasizes performance issues.

Built-In Collections: Lists, Tuples, Sets, Dictionaries

- There's little reason today for most application developers to build *custom* data structures. This is a subject for CS2 courses—our scope is *strictly* CS1 and the corresponding data science course(s). The book features a solid **two-chapter**

12. `https://public.dhe.ibm.com/common/ssi/ecm/wr/en/wrl12345usen/watson-customer-engagement-watson-marketing-wr-other-papers-and-reports-wrl12345usen-20170719.pdf`.
13. `https://analyticsweek.com/content/big-data-facts/`.

treatment of Python's built-in data structures—lists, **tuples**, **dictionaries** and **sets—**with which most data-structuring tasks can be accomplished.

Array-Oriented Programming with NumPy Arrays and Pandas Series/DataFrames

- We take an innovative approach in this book by focusing on three key data structures from open-source libraries—NumPy arrays, pandas `Series` and pandas `DataFrames`. These libraries are used extensively in data science, computer science, artificial intelligence and big data. NumPy offers as much as two orders of magnitude higher performance than built-in Python lists.
- We include in Chapter 7 a rich treatment of NumPy arrays. Many libraries, such as pandas, are built on NumPy. The **Intro to Data Science sections** in Chapters 7–9 introduce pandas `Series` and `DataFrames`, which along with NumPy arrays are then used throughout the remaining chapters.

File Processing and Serialization

- Chapter 9 presents **text-file processing**, then demonstrates how to serialize objects using the popular **JSON (JavaScript Object Notation)** format. JSON is a commonly used data-interchange format that you'll frequently see used in the data science chapters—often with libraries that hide the JSON details for simplicity.
- Many data science libraries provide built-in file-processing capabilities for loading datasets into your Python programs. In addition to plain text files, we process files in the popular **CSV (comma-separated values) format** using the Python Standard Library's `csv` module and capabilities of the pandas data science library.

Object-Based Programming

- In all the Python code we studied during our research for this book, we rarely encountered *custom classes*. These are common in the powerful libraries *used* by Python programmers.
- We emphasize using the enormous number of valuable classes that the **Python open-source community** has packaged into industry standard class libraries. You'll focus on knowing what libraries are out there, choosing the ones you'll need for your app, creating objects from existing classes (usually in one or two lines of code) and making them "jump, dance and sing." This is called **object-*based* programming**—it enables you to **build impressive applications concisely**, which is a significant part of Python's appeal.
- With this approach, you'll be able to use machine learning, deep learning, reinforcement learning (in the exercises) and other AI technologies to solve a wide range of intriguing problems, including **cognitive computing** challenges like **speech recognition** and **computer vision**. In the past, with just an introductory programming course, you never would have been able to tackle such tasks.

Object-Oriented Programming

- For computer science students, developing *custom* classes is a crucial **object-oriented programming** skill, along with inheritance, polymorphism and duck typing. We discuss these in Chapter 10.

- The object-oriented programming treatment is modular, so instructors can present basic or intermediate coverage.
- Chapter 10 includes a discussion of unit testing with `doctest` and a fun card-shuffling-and-dealing simulation.
- The six data science, AI, big data and cloud chapters require only a few straightforward custom class definitions. Instructors who do not wish to cover Chapter 10 can have students simply mimic our class definitions.

Privacy

- In the exercises, you'll research ever-stricter privacy laws such as **HIPAA** (**Health Insurance Portability and Accountability Act**) in the United States and **GDPR** (**General Data Protection Regulation**) for the European Union. A key aspect of privacy is protecting users' **personally identifiable information** (**PII**), and a key challenge with big data is that it's easy to cross-reference facts about individuals among databases. We mention privacy issues in several places throughout the book.

Security

- Security is crucial to privacy. We deal with some Python-specific security issues.
- AI and big data present unique privacy, security and ethical challenges. In the exercises, students will research the **OWASP Python Security Project** (`http://www.pythonsecurity.org/`), **anomaly detection**, **blockchain** (the technology behind cryptocurrencies like BitCoin and Ethereum) and more.

Ethics

- Ethics conundrum: Suppose big data analytics with AI predicts that a person with no criminal record has a significant chance of committing a serious crime. Should that person be arrested? In the exercises, you'll research this and other ethical issues, including *deep fakes* (AI-generated images and videos that appear to be real), *bias* in machine learning and *CRISPR gene editing*. Students also investigate privacy and ethical issues surrounding AIs and **intelligent assistants**, such as **IBM Watson**, **Amazon Alexa**, **Apple Siri**, **Google Assistant** and **Microsoft Cortana**. For example, just recently, a judge ordered Amazon to turn over Alexa recordings for use in a criminal case.[14]

Reproducibility

- In the sciences in general, and data science in particular, there's a need to reproduce the results of experiments and studies, and to communicate those results effectively. **Jupyter Notebooks** are a preferred means for doing this.
- We provide you with a Jupyter Notebooks experience to help meet the reproducibility recommendations of the data science undergraduate curriculum proposal.
- We discuss *reproducibility* throughout the book in the context of programming techniques and software such as Jupyter Notebooks and **Docker**.

14. `https://techcrunch.com/2018/11/14/amazon-echo-recordings-judge-murder-case/`.

Transparency

- The data science curriculum proposal mentions data transparency. One aspect of data transparency is the availability of data. Many governments and other organization now adhere to **open-data** principles, enabling anyone to access their data.[15] We point you to a wide range of datasets that are made available by such entities.
- Other aspects of data transparency include determining that data is correct and knowing its origin (think, for example, of "fake news"). Many of the datasets we use are bundled with key libraries we present, such as **Scikit-learn** for machine learning and **Keras** for deep learning. We also point you to various curated **dataset repositories** such as the **University of California Irvine** (UCI) **Machine Learning Repository** (with 450+ datasets)[16] and **Carnegie Mellon University's StatLib Datasets Archive** (with 100+ datasets).[17]

Performance

- We use the **`timeit` profiling tool** in several examples and exercises to compare the performance of different approaches to performing the same tasks. Other performance-related discussions include generator expressions, NumPy arrays vs. Python lists, performance of machine-learning and deep-learning models, and Hadoop and Spark distributed-computing performance.

Big Data and Parallelism

- Computer applications have generally been good at doing one thing at a time. Today's more sophisticated applications need to be able to do many things in parallel. The human brain is believed to have the equivalent of 100 billion parallel processors.[18] For years we've written about parallelism at the program level, which is complex and error-prone.
- In this book, rather than writing your own parallelization code, you'll let libraries like Keras running over TensorFlow, and big data tools like Hadoop and Spark parallelize operations for you. In this big data/AI era, the sheer processing requirements of massive data apps demand taking advantage of true parallelism provided by **multicore processors**, **graphics processing units** (GPUs), **tensor processing units** (TPUs) and huge ***clusters* of computers in the cloud**. Some big data tasks could have thousands of processors working in parallel to analyze massive amounts of data in reasonable time. Sequentializing such processing is typically not an option, because it would take too long.

15. `https://www.mckinsey.com/~/media/McKinsey/Business%20Functions/McKinsey%20Digital/Our%20Insights/Big%20data%20The%20next%20frontier%20for%20innovation/MGI_big_data_full_report.ashx` (page 56).
16. `https://archive.ics.uci.edu/ml/datasets.html`.
17. `http://lib.stat.cmu.edu/datasets/`.
18. `https://www.technologyreview.com/s/532291/fmri-data-reveals-the-number-of-parallel-processes-running-in-the-brain/`.

Chapter Dependencies

If you're an instructor planning your course syllabus or a professional deciding which chapters to read, this section will help you make the best decisions. Please read the one-page **Table of Contents** on the first page of the book—this will quickly familiarize you with the book's unique architecture. Teaching or reading the chapters in order is easiest. However, much of the content in the Intro to Data Science sections at the ends of Chapters 1–10 and the case studies in Chapters 12–17 requires only Chapters 1–5 and small portions of Chapters 6–10 as discussed below.

Part 1: Python Fundamentals Quickstart

We recommend that all courses cover Python Chapters 1–5:

- **Chapter 1, Introduction to Computers and Python**, introduces concepts that lay the groundwork for the Python programming in Chapters 2–11 and the big data, artificial-intelligence and cloud-based case studies in Chapters 12–17. The chapter also includes **test-drives** of **IPython** and **Jupyter Notebooks**.
- **Chapter 2, Introduction to Python Programming,** presents Python programming fundamentals with code examples illustrating key language features.
- **Chapter 3, Control Statements and Program Development**, presents Python's **control statements**, focuses on **problem-solving and algorithm development**, and introduces **basic list processing**.
- **Chapter 4, Functions**, introduces program construction using existing functions and custom functions as building blocks, presents **simulation techniques** with **random-number generation** and introduces **tuple fundamentals**.
- **Chapter 5, Sequences: Lists and Tuples**, presents Python's built-in list and tuple collections in more detail and begins our introduction to **functional-style programming**.

Part 2: Python Data Structures, Strings and Files[19]

The following summarizes inter-chapter dependencies for Python Chapters 6–9 and assumes that you've read Chapters 1–5.

- **Chapter 6, Dictionaries and Sets**—The Intro to Data Science section is not dependent on Chapter 6's contents.
- **Chapter 7, Array-Oriented Programming with NumPy**—The Intro to Data Science section requires dictionaries (Chapter 6) and arrays (Chapter 7).
- **Chapter 8, Strings: A Deeper Look**—The Intro to Data Science section requires raw strings and regular expressions (Sections 8.11–8.12), and pandas `Series` and `DataFrame` features from Section 7.14's Intro to Data Science.
- **Chapter 9, Files and Exceptions**—For **JSON serialization**, it's useful to understand dictionary fundamentals (Section 6.2). Also, the Intro to Data Science section requires the built-in `open` function and the `with` statement (Section 9.3), and pandas `DataFrame` features from Section 7.14's Intro to Data Science.

19. We could have included Chapter 5 in Part 2. We placed it in Part 1 because that's the group of chapters all courses should cover.

Part 3: Python High-End Topics

The following summarizes inter-chapter dependencies for Python Chapters 10–11 and assumes that you've read Chapters 1–5.

- **Chapter 10, Object-Oriented Programming**—The Intro to Data Science requires pandas `DataFrame` features from the Intro to Data Science Section 7.14. Instructors wanting to cover only **classes and objects** can present Sections 10.1–10.6. Instructors wanting to cover more advanced topics like **inheritance, polymorphism and duck typing**, can present Sections 10.7–10.9. Sections 10.10–10.15 provide additional advanced perspectives.
- **Chapter 11, Computer Science Thinking: Recursion, Searching, Sorting and Big O**—Requires creating and accessing the elements of arrays (Chapter 7), the `%timeit` magic (Section 7.6), string method `join` (Section 8.9) and Matplotlib `FuncAnimation` from Section 6.4's Intro to Data Science.

Part 4: AI, Cloud and Big Data Case Studies

The following summary of inter-chapter dependencies for Chapters 12–17 assumes that you've read Chapters 1–5. Most of Chapters 12–17 also require dictionary fundamentals from Section 6.2.

- **Chapter 12, Natural Language Processing** (NLP), uses pandas `DataFrame` features from Section 7.14's Intro to Data Science.
- **Chapter 13, Data Mining Twitter**, uses pandas `DataFrame` features from Section 7.14's Intro to Data Science, string method `join` (Section 8.9), JSON fundamentals (Section 9.5), `TextBlob` (Section 12.2) and Word clouds (Section 12.3). Several examples require defining a class via inheritance (Chapter 10), but readers can simply mimic the class definitions we provide without reading Chapter 10.
- **Chapter 14, IBM Watson and Cognitive Computing**, uses built-in function `open` and the `with` statement (Section 9.3).
- **Chapter 15, Machine Learning: Classification, Regression and Clustering**, uses NumPy array fundamentals and method `unique` (Chapter 7), pandas `DataFrame` features from Section 7.14's Intro to Data Science and Matplotlib function `subplots` (Section 10.6).
- **Chapter 16, Deep Learning**, requires NumPy array fundamentals (Chapter 7), string method `join` (Section 8.9), general machine-learning concepts from Chapter 15 and features from Chapter 15's Case Study: Classification with k-Nearest Neighbors and the Digits Dataset.
- **Chapter 17, Big Data: Hadoop, Spark, NoSQL and IoT**, uses string method `split` (Section 6.2.7), Matplotlib `FuncAnimation` from Section 6.4's Intro to Data Science, pandas `Series` and `DataFrame` features from Section 7.14's Intro to Data Science, string method `join` (Section 8.9), the `json` module (Section 9.5), NLTK stop words (Section 12.2.13) and from Chapter 13 Twitter authentication, Tweepy's `StreamListener` class for streaming tweets, and the `geopy` and `folium` libraries. A few examples require defining a class via inheritance (Chapter 10), but readers can simply mimic the class definitions we provide without reading Chapter 10.

Computing and Data Science Curricula

We read the following ACM/IEEE CS-and-related curriculum documents in preparation for writing this book:

- Computer Science Curricula 2013,[20]
- CC2020: A Vision on Computing Curricula,[21]
- Information Technology Curricula 2017,[22]
- Cybersecurity Curricula 2017,[23]

and the 2016 data science initiative "**Curriculum Guidelines for Undergraduate Programs in Data Science**"[24] from the faculty group sponsored by the NSF and the Institute for Advanced Study.

Computing Curricula

- According to "CC2020: A Vision on Computing Curricula," the curriculum "needs to be reviewed and updated to include the new and emerging areas of computing such as **cybersecurity** and **data science**."[25]
- Data science includes key topics (besides general-purpose programming) such as machine learning, deep learning, natural language processing, speech synthesis and recognition and others that are classic artificial intelligence (AI)—and hence CS topics as well.

Data Science Curriculum

- Graduate-level data science is well established and the undergraduate level is growing rapidly to meet strong industry demand. Our hands-on, nonmathematical, project-oriented, programming-intensive approach facilitates moving data science into the undergraduate curriculum, based on the proposed new curriculum.
- There already are lots of undergraduate data science and data analytics programs, but they're not uniform. That was part of the motivation for the 25 faculty members on the data science curriculum committee to get together in 2016 and develop the proposed 10-course undergraduate major in data science, "Curriculum Guidelines for Undergraduate Programs in Data Science."
- The curriculum committee says that "many of the courses traditionally found in computer science, statistics, and mathematics offerings should be redesigned for

20. ACM/IEEE (Assoc. Comput. Mach./Inst. Electr. Electron. Eng.). 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science* (New York: ACM), `http://ai.stanford.edu/users/sahami/CS2013/final-draft/CS2013-final-report.pdf`.
21. A. Clear, A. Parrish, G. van der Veer and M. Zhang "CC2020: A Vision on Computing Curricula," `https://dl.acm.org/citation.cfm?id= 3017690`.
22. *Information Technology Curricula 2017*, `http://www.acm.org/binaries/content/assets/education/it2017.pdf`.
23. *Cybersecurity Curricula 2017*, `https://cybered.hosting.acm.org/wp-content/uploads/2018/02/newcover_csec2017.pdf`.
24. "Curriculum Guidelines for Undergraduate Programs in Data Science," `http://www.annualreviews.org/doi/full/10.1146/annurev-statistics-060116-053930`.
25. `http://delivery.acm.org/10.1145/3020000/3017690/p647-clear.pdf`.

the data science major in the interests of efficiency and the potential synergy that integrated courses would offer."[26]

- The committee recommends integrating these areas with computational and statistical thinking in all courses, and indicates that ***new textbooks will be essential***[27]**—this book is designed with the committee's recommendations in mind**.
- Python has rapidly become one of the world's most popular general-purpose programming languages. For schools that want to **cover only one language** in their data science major, it's reasonable that Python be that language.

Data Science Overlaps with Computer Science[28]

The undergraduate data science curriculum proposal includes algorithm development, programming, computational thinking, data structures, database, mathematics, statistical thinking, machine learning, data science and more—a significant overlap with computer science, especially given that the data science courses include some key AI topics. Even though ours is a Python programming textbook, it touches each of these areas (except for heavy mathematics) from the recommended data science 10-course curriculum, as we efficiently work data science into various examples, exercises, projects and full-implementation case studies.

Key Points from the Data Science Curriculum Proposal

In this section, we call out some key points from the data science undergraduate curriculum proposal[29] or its detailed course descriptions appendix.[30] We worked hard to incorporate these and many other objectives:

- learn **programming fundamentals** commonly presented in **computer science** courses, including working with **data structures**.
- be able to **solve problems** by **creating algorithms**.
- work with **procedural**, **functional** and **object-oriented programming**.
- receive an integrated presentation of **computational and statistical thinking**, including **exploring concepts via simulations**.
- use **development environments** (we use **IPython** and **Jupyter Notebooks**).
- work with **real-world data** in **practical case studies** and **projects in every course**.
- **obtain**, **explore** and **transform (wrangle) data** for analysis.
- create **static**, **dynamic** and **interactive data visualizations**.

---

26. "Curriculum Guidelines for Undergraduate Programs in Data Science," `http://www.annualreviews.org/doi/full/10.1146/annurev-statistics-060116-053930` (pp. 16–17).
27. "Curriculum Guidelines for Undergraduate Programs in Data Science," `http://www.annualreviews.org/doi/full/10.1146/annurev-statistics-060116-053930` (pp. 16–17).
28. This section is intended primarily for data science instructors. Given that the emerging 2020 Computing Curricula for computer science and related disciplines is likely to include some key data science topics, this section includes important information for computer science instructors as well.
29. "Curriculum Guidelines for Undergraduate Programs in Data Science," `http://www.annualreviews.org/doi/full/10.1146/annurev-statistics-060116-053930`.
30. "Appendix—Detailed Courses for a Proposed Data Science Major," `http://www.annualreviews.org/doi/suppl/10.1146/annurev-statistics-060116-053930/suppl_file/st04_de_veaux_supmat.pdf`.

- communicate **reproducible results**.
- work with **existing software** and **cloud-based tools**.
- work with **statistical and machine-learning models**.
- work with **high-performance tools** (**Hadoop**, **Spark**, **MapReduce** and **NoSQL**).
- focus on **data's ethics**, **security**, **privacy**, **reproducibility** and **transparency** issues.

Jobs Requiring Data Science Skills

In 2011, McKinsey Global Institute produced their report, "Big data: The next frontier for innovation, competition and productivity." In it, they said, "The United States alone faces a shortage of 140,000 to 190,000 people with deep analytical skills as well as 1.5 million managers and analysts to analyze big data and make decisions based on their findings."[31] This continues to be the case. The August 2018 "LinkedIn Workforce Report" says the United States has a shortage of over 150,000 people with data science skills.[32] A 2017 report from IBM, Burning Glass Technologies and the Business-Higher Education Forum, says that by 2020 in the United States there will be hundreds of thousands of new jobs requiring data science skills.[33]

Jupyter Notebooks

For your convenience, we provide the book's examples in **Python source code** (**`.py`**) **files** for use with the command-line IPython interpreter *and* as **Jupyter Notebooks** (**`.ipynb`**) **files** that you can load into your web browser and execute. You can use whichever method of executing code examples you prefer.

Jupyter Notebooks is a free, open-source project that enables authors to combine text, graphics, audio, video, and interactive coding functionality for entering, editing, executing, debugging, and modifying code quickly and conveniently in a web browser. According to the article, "What Is Jupyter?":

> *Jupyter has become a standard for scientific research and data analysis. It packages computation and argument together, letting you build "computational narratives"; … and it simplifies the problem of distributing working software to teammates and associates.*[34]

In our experience, it's a wonderful learning environment and **rapid prototyping tool** for novices and experienced developers alike. For this reason, we use **Jupyter Notebooks** rather than a traditional **integrated development environment** (**IDE**), such as **Eclipse**, **Visual Studio**, **PyCharm** or **Spyder**. Academics and professionals already use Jupyter extensively for sharing research results. Jupyter Notebooks support is provided through the traditional open-source community mechanisms[35] (see "Getting Your Questions Answered" later in this Preface).

31. `https://www.mckinsey.com/~/media/McKinsey/Business%20Functions/McKinsey%20Digital/Our%20Insights/Big%20data%20The%20next%20frontier%20for%20innovation/MGI_big_data_full_report.ashx` (page 3).
32. `https://economicgraph.linkedin.com/resources/linkedin-workforce-report-august-2018`.
33. `https://www.burning-glass.com/wp-content/uploads/The_Quant_Crunch.pdf` (page 3).
34. `https://www.oreilly.com/ideas/what-is-jupyter`.
35. `https://jupyter.org/community`.

We believe Jupyter Notebooks are a compelling way to teach and learn Python and that most instructors will choose to use Jupyter. The notebooks include:

- examples,
- Self Check exercises,
- all end-of-chapter exercises containing code, such as **"What does this code do?"** and **"What's wrong with this code?" exercises**.
- **Visualizations and animations**, which are a crucial part of the book's pedagogy. We provide the code in Jupyter Notebooks so students can conveniently **reproduce our results**.

See the Before You Begin section that follows this Preface for software installation details and see the test-drives in Section 1.10 for information on running the book's examples.

Collaboration and Sharing Results

Working in teams and **communicating research results** are both emphasized in the proposed undergraduate data science curriculum[36] and are important for students moving into data-analytics positions in industry, government or academia:

- The notebooks you create are **easy to share** among team members simply by copying the files or via **GitHub**.
- Research results, including code and insights, can be shared as static web pages via tools like **nbviewer** (`https://nbviewer.jupyter.org`) and **GitHub**—both automatically render notebooks as web pages.

Reproducibility: A Strong Case for Jupyter Notebooks

In data science, and in the sciences in general, experiments and studies should be **reproducible**. This has been written about in the literature for many years, including

- Donald Knuth's 1992 computer science publication—*Literate Programming*.[37]
- The article "Language-Agnostic Reproducible Data Analysis Using Literate Programming,"[38] which says, "Lir (literate, reproducible computing) is based on the idea of literate programming as proposed by Donald Knuth."

Essentially, reproducibility captures the complete environment used to produce results—hardware, software, communications, algorithms (especially code), data and the **data's provenance** (origin and lineage).

The undergraduate data science curriculum proposal mentions reproducibility as a goal in four places. The article "50 Years of Data Science" says, "teaching students to work reproducibly enables easier and deeper evaluation of their work; having them reproduce parts of analyses by others allows them to learn skills like **Exploratory Data Analysis** that are commonly practiced but not yet systematically taught; and training them to work reproducibly will make their post-graduation work more reliable."[39]

36. "Curriculum Guidelines for Undergraduate Programs in Data Science," `http://www.annualreviews.org/doi/full/10.1146/annurev-statistics-060116-053930` (pp. 18–19).
37. Knuth, D., "Literate Programming" (PDF), *The Computer Journal*, British Computer Society, 1992.
38. `http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0164023`.

Docker

In Chapter 17, we'll introduce **Docker**—a tool for packaging software into *containers* that bundle *everything* required to execute that software conveniently, reproducibly and portably across platforms. Some software packages we use in Chapter 17 require complicated setup and configuration. For many of these, you can download free preexisting **Docker containers**. These enable you to avoid complex installation issues and execute software locally on your desktop or notebook computers, making Docker a great way to help you get started with new technologies quickly and conveniently.

Docker also helps with **reproducibility**. You can create custom Docker containers that are configured with the versions of every piece of software and every library you used in your study. This would enable others to recreate the environment you used, then reproduce your work, and will help you reproduce your own results. In Chapter 17, you'll use Docker to download and execute a container that's preconfigured for you to code and run big data Spark applications using Jupyter Notebooks.

Class Tested

While the book was under development, one of our academic reviewers—Dr. Alison Sanchez, Assistant Professor in Economics, University of San Diego—class tested it in a new course, "Business Analytics Strategy." She commented: "Wonderful for first-time Python learners from all educational backgrounds and majors. My business analytics students had little to no coding experience when they began the course. In addition to loving the material, it was easy for them to follow along with the example exercises and by the end of the course were able to mine and analyze Twitter data using techniques learned from the book. The chapters are clearly written with detailed explanations of the example code—which makes it easy for students without a computer science background to understand. The modular structure, wide range of contemporary data science topics, and companion Jupyter notebooks make this a fantastic resource for instructors and students of a variety of Data Science, Business Analytics and Computer Science courses."

"Flipped Classroom"

Many instructors are now using "flipped classrooms."[40,41] Students learn the content on their own before coming to class (typically via video lectures), and class time is used for tasks such as hands-on coding, working in groups and discussions. Our book and supplements are appropriate for flipped classrooms:

- We provide extensive VideoNotes in which co-author Paul Deitel teaches the concepts in the core Python chapters. See "Student and Instructor Supplements" later in this Preface for details on accessing the videos.
- Some students learn best by —and video is *not* hands-on. **One of the most compelling features of the book** is its interactive approach with **538 Python code**

39. "50 Years of Data Science," `http://courses.csail.mit.edu/18.337/2015/docs/50YearsDataScience.pdf`, p. 33.
40. `https://en.wikipedia.org/wiki/Flipped_classroom`.
41. `https://www.edsurge.com/news/2018-05-24-a-case-for-flipping-learning-without-videos`.

examples—many with just one or a few snippets—and 557 **Self Check exercises with answers**. These enable students to learn in small pieces with immediate feedback—perfect for active self-paced learning. Students can easily modify the "hot" code and see the effects of their changes.

- Our **Jupyter Notebooks supplements** provide a convenient mechanism for students to work with the code.
- We provide 471 exercises and projects, which students can work on at home and/or in class. Many of these are appropriate for group projects.
- We provide lots of probing questions on ethics, privacy, security and more in the exercises and projects. These are appropriate for in-class discussions and group work.

Special Feature: IBM Watson Analytics and Cognitive Computing

Early in our research for this book, we recognized the rapidly growing interest in **IBM's Watson**. We investigated competitive services and found Watson's "no credit card required" policy for its "free tiers" to be among the most friendly for our readers.

IBM Watson is a **cognitive-computing** platform being employed across a wide range of real-world scenarios. Cognitive-computing systems simulate the **pattern-recognition** and **decision-making** capabilities of the human brain to "learn" as they consume more data.[42,43,44] We include a significant hands-on Watson treatment. We use the free **Watson Developer Cloud: Python SDK**, which provides application programming interfaces (APIs) that enable you to interact with Watson's services programmatically. Watson is fun to use and a great platform for letting your creative juices flow. You'll demo or use the following Watson APIs: **Conversation**, **Discovery**, **Language Translator**, **Natural Language Classifier**, **Natural Language Understanding**, **Personality Insights**, **Speech to Text**, **Text to Speech**, **Tone Analyzer** and **Visual Recognition**.

Watson's Lite Tier Services and Watson Case Study

IBM encourages learning and experimentation by providing *free lite tiers* for many of their APIs.[45] In Chapter 14, you'll try demos of many Watson services.[46] Then, you'll use the lite tiers of Watson's **Text to Speech**, **Speech to Text** and **Translate** services to implement a **"traveler's assistant" translation app**. You'll speak a question in English, then the app will transcribe your speech to English text, translate the text to Spanish and speak the Spanish text. Next, you'll speak a Spanish response (in case you don't speak Spanish, we provide an audio file you can use). Then, the app will quickly transcribe the speech to Spanish text, translate the text to English and speak the English response. Cool stuff!

42. `http://whatis.techtarget.com/definition/cognitive-computing`.
43. `https://en.wikipedia.org/wiki/Cognitive_computing`.
44. `https://www.forbes.com/sites/bernardmarr/2016/03/23/what-everyone-should-know-about-cognitive-computing`.
45. Always check the latest terms on IBM's website, as the terms and services may change.
46. `https://console.bluemix.net/catalog/`.

Teaching Approach

Intro to Python for Computer Science and Data Science contains a rich collection of examples, exercises and projects drawn from many fields. Students solve interesting, real-world problems working with **real-world datasets**. The book concentrates on the principles of good **software engineering** and stresses **program clarity**.

Using Fonts for Emphasis

We place the key terms and the index's page reference for each defining occurrence in **bold** text for easier reference. We place on-screen components in the **bold Helvetica** font (for example, the **File** menu) and use the `Lucida` font for Python code (for example, `x = 5`).

Syntax Coloring

The book is in full color. For readability, we syntax color all the code. Our syntax-coloring conventions are as follows:

```
comments appear in green
keywords appear in dark blue
constants and literal values appear in light blue
errors appear in red
all other code appears in black
```

Objectives and Outline

Each chapter begins with objectives that tell you what to expect and give you an opportunity, after reading the chapter, to determine whether it has met the intended goals. The chapter outline enables students to approach the material in top-down fashion.

538 Examples

The book's **538 examples** contain approximately **4000 lines of code**. This is a relatively small amount of code for a book this size and is due to the fact that Python is such an expressive language. Also, our coding style is to use powerful class libraries to do most of the work wherever possible.

160 Tables/Illustrations/Visualizations

Abundant tables, line drawings, and visualizations are included. The visualizations are in color and include some 2D and 3D, some static and dynamic and some interactive.

Programming Wisdom

We integrate into the discussions programming wisdom from the authors' combined nine decades of programming and teaching experience, including:

- **Good programming practices** and preferred Python idioms that help you produce clearer, more understandable and more maintainable programs.
- **Common programming errors** to reduce the likelihood that you'll make them.
- **Error-prevention tips** with suggestions for exposing bugs and removing them from your programs. Many of these tips describe techniques for preventing bugs from getting into your programs in the first place.
- **Performance tips** that highlight opportunities to make your programs run faster or minimize the amount of memory they occupy.

- **Software engineering observations** that highlight architectural and design issues for proper software construction, especially for larger systems.

Wrap-Up
Chapters 2–17 end with Wrap-Up sections summarizing what you've learned.

Index
We have included an extensive index. The defining occurrences of key terms are highlighted with a **bold** page number.

Software Used in the Book
All the software you'll need for this book is available for Windows, macOS and Linux and is free for download from the Internet. We wrote the book's examples using the free **Anaconda Python distribution**. It includes most of the Python, visualization and data science libraries you'll need, as well as Python, the IPython interpreter, Jupyter Notebooks and Spyder, considered one of the best Python data science integrated development environments (IDEs)—we use only IPython and Jupyter Notebooks for program development in the book. The Before You Begin section discusses installing Anaconda and other items you'll need for working with our examples.

Python Documentation
You'll find the following documentation especially helpful as you work through the book:

- The Python Standard Library:

  `https://docs.python.org/3/library/index.html`
- The Python Language Reference:

  `https://docs.python.org/3/reference/index.html`
- Python documentation list:

  `https://docs.python.org/3/`

Getting Your Questions Answered
Online forums enable you to interact with other Python programmers and get your Python questions answered. Popular Python and general programming forums include:

- `python-forum.io`
- `StackOverflow.com`
- `https://www.dreamincode.net/forums/forum/29-python/`

Also, many vendors provide forums for their tools and libraries. Most of the libraries you'll use in this book are managed and maintained at `github.com`. Some library maintainers provide support through the **Issues** tab on a given library's GitHub page. If you cannot find an answer to your questions online, please see our web page for the book at

`http://www.deitel.com`[47]

47. Our website is undergoing a major upgrade. If you do not find something you need, please write to us directly at `deitel@deitel.com`.

Getting Jupyter Help

Jupyter Notebooks support is provided through:

- Project Jupyter Google Group:

  `https://groups.google.com/forum/#!forum/jupyter`

- Jupyter real-time chat room:

  `https://gitter.im/jupyter/jupyter`

- GitHub

  `https://github.com/jupyter/help`

- StackOverflow:

  `https://stackoverflow.com/questions/tagged/jupyter`

- Jupyter for Education Google Group (for instructors teaching with Jupyter):

  `https://groups.google.com/forum/#!forum/jupyter-education`

Student and Instructor Supplements

The following supplements are available to students and instructors.

Code Examples, Videos Notes and Companion Website

To get the most out of the presentation, you should execute each code example in parallel with reading the corresponding discussion. The book's **Companion Website** at

`https://www.pearsonglobaleditions.com`

contains:

- **Downloadable Python source code** (`.py` files) and **Jupyter Notebooks** (`.ipynb` files) for the book's **code examples**, for code-based **Self-Check Exercises** and for **end-of-chapter exercises** that have code as part of the exercise description.
- **VideoNotes,** in which co-author Paul Deitel explains most of the examples in the book's core Python chapters.

For download instructions, see the *Before You Begin* section that follows this Preface. New copies of this book come with a **Companion Website access code** on the book's inside front cover. If the access code is already visible or there isn't one, you purchased a used book. Access codes are also available on the inside front cover if you are using the eBook or the eText.

Instructor Supplements on Pearson's Instructor Resource Center

The following supplements are available to qualified instructors only through Pearson Education's IRC (Instructor Resource Center) at `https://www.pearsonglobaleditions.com`:

- **PowerPoint slides.**
- **Instructor Solutions Manual** with solutions to many of the exercises. Solutions are *not* provided for "project" and "research" exercises—many of which are substantial and appropriate for term projects, directed-study projects, capstone-course

projects and thesis topics. **Before assigning a particular exercise for homework, instructors should check the IRC to be sure the solution is available.**

- **Test Item File** with multiple-choice, short-answer questions and answers. These are easy to use in automated assessment tools.

Please do not write to us requesting access to the Pearson Instructor's Resource Center which contains the book's instructor supplements, including exercise solutions. Access is strictly limited to college instructors teaching from the book. Instructors may obtain access through their Pearson representatives.

Keeping in Touch with the Authors

For answers to questions, syllabus assistance or to report an error, send an e-mail to us at

`deitel@deitel.com`

or interact with us via **social media**:

- **Facebook**® (`http://www.deitel.com/deitelfan`)
- **Twitter**® (`@deitel`)
- **LinkedIn**® (`http://linkedin.com/company/deitel-&-associates`)
- **YouTube**® (`http://youtube.com/DeitelTV`)

Acknowledgments

We'd like to thank Barbara Deitel for long hours devoted to Internet research on this project. We're fortunate to have worked with the dedicated team of publishing professionals at Pearson. We appreciate the guidance, wisdom and energy of Tracy Johnson (Executive Portfolio Manager, Higher Ed Courseware, Computer Science)—she challenged us at every step of the process to "get it right." Carole Snyder managed the book's production and interacted with Pearson's permissions team, promptly clearing our graphics and citations.

We wish to acknowledge the efforts of our academic and professional reviewers. Meghan Jacoby and Patricia Byron-Kimball recruited the reviewers and managed the review process.

Adhering to a tight schedule, the reviewers scrutinized our work, providing countless suggestions for improving the accuracy, completeness and timeliness of the presentation.

Reviewers

Proposal Reviewers

Dr. Irene Bruno, Associate Professor in the Department of Information Sciences and Technology, George Mason University
Lance Bryant, Associate Professor, Department of Mathematics, Shippensburg University
Daniel Chen, Data Scientist, Lander Analytics
Garrett Dancik, Associate Professor of Computer Science/Bioinformatics Department of Computer Science, Eastern Connecticut State University
Dr. Marsha Davis, Department Chair of Mathematical Sciences, Eastern Connecticut State University
Roland DePratti, Adjunct Professor of Computer Science, Eastern Connecticut State University
Shyamal Mitra, Senior Lecturer, Computer Science, University of Texas at Austin
Dr. Mark Pauley, Senior Research Fellow, Bioinformatics, School of Interdisciplinary Informatics, University of Nebraska at Omaha
Sean Raleigh, Associate Professor of Mathematics, Chair of Data Science, Westminster College
Alison Sanchez, Assistant Professor in Economics, University of San Diego
Dr. Harvey Siy, Associate Professor of Computer Science, Information Science and Technology, University of Nebraska at Omaha
Jamie Whitacre, Independent Data Science Consultant

Book Reviewers

Daniel Chen, Data Scientist, Lander Analytics
Garrett Dancik, Associate Professor of Computer Science/Bioinformatics, Eastern Connecticut State University
Pranshu Gupta, Assistant Professor, Computer Science, DeSales University
David Koop, Assistant Professor, Data Science Program Co-Director, U-Mass Dartmouth
Ramon Mata-Toledo, Professor, Computer Science, James Madison University
Shyamal Mitra, Senior Lecturer, Computer Science, University of Texas at Austin
Alison Sanchez, Assistant Professor in Economics, University of San Diego
José Antonio González Seco, IT Consultant
Jamie Whitacre, Independent Data Science Consultant
Elizabeth Wickes, Lecturer, School of Information Sciences, University of Illinois

A Special Thank You

Our thanks to Prof. Alison Sanchez for class-testing the book prepublication in her new “Business Analytics Strategy” class at the University of San Diego. She reviewed the lengthy proposal, adopting the book sight unseen and signed on as a full-book reviewer in parallel with using the book in her class. Her guidance (and courage) throughout the entire book-development process are sincerely appreciated.

Well, there you have it! As you read the book, we’d appreciate your comments, criticisms, corrections and suggestions for improvement. Please send all correspondence to:

```
deitel@deitel.com
```

We’ll respond promptly.

Welcome again to the exciting open-source world of Python programming. We hope you enjoy this look at leading-edge computer-applications development with Python, IPython, Jupyter Notebooks, AI, big data and the cloud. We wish you great success!

Paul and Harvey Deitel

Acknowledgments for the Global Edition

Pearson would like to acknowledge and thank the following for their work on the Global Edition.

Contributor
Greet Baldewijns, KU Leuven

Reviewers
Ritesh Ajoodha, The University of the Witwatersrand
David Merand, The University of the Witwatersrand
Faruk Tokdemir, Middle East Technical University

About the Authors

Paul J. Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., is an MIT graduate with 38 years of experience in computing. Paul is one of the world's most experienced programming-languages trainers, having taught professional courses to software developers since 1992. He has delivered hundreds of programming courses to industry clients internationally, including Cisco, IBM, Siemens, Sun Microsystems (now Oracle), Dell, Fidelity, NASA at the Kennedy Space Center, the National Severe Storm Laboratory, White Sands Missile Range, Rogue Wave Software, Boeing, Nortel Networks, Puma, iRobot and many more. He and his co-author, Dr. Harvey M. Deitel, are the world's best-selling programming-language textbook/professional book/video authors.

Dr. Harvey M. Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has 58 years of experience in computing. Dr. Deitel earned B.S. and M.S. degrees in Electrical Engineering from MIT and a Ph.D. in Mathematics from Boston University—he studied computing in each of these programs before they spun off Computer Science programs. He has extensive college teaching experience, including earning tenure and serving as the Chairman of the Computer Science Department at Boston College before founding Deitel & Associates, Inc., in 1991 with his son, Paul. The Deitels' publications have earned international recognition, with more than 100 translations published in Japanese, German, Russian, Spanish, French, Polish, Italian, Simplified Chinese, Traditional Chinese, Korean, Portuguese, Greek, Urdu and Turkish. Dr. Deitel has delivered hundreds of programming courses to academic, corporate, government and military clients.

About Deitel® & Associates, Inc.

Deitel & Associates, Inc., founded by Paul Deitel and Harvey Deitel, is an internationally recognized authoring and corporate training organization, specializing in computer programming languages, object technology, mobile app development and Internet and web software technology. The company's training clients include some of the world's largest companies, government agencies, branches of the military, and academic institutions. The company offers instructor-led training courses delivered at client sites worldwide on major programming languages and platforms.

Through its 44-year publishing partnership with Pearson/Prentice Hall, Deitel & Associates, Inc., publishes leading-edge programming textbooks and professional books in print and e-book formats, **LiveLessons** video courses, Safari-Live online seminars and **Revel™** interactive multimedia courses. To contact Deitel & Associates, Inc. and the authors, or to request a proposal on-site, instructor-led training, write to:

`deitel@deitel.com`

To learn more about Deitel on-site corporate training, visit

`http://www.deitel.com/training`

Before You Begin

This section contains information you should review before using this book.

Font and Naming Conventions

We show Python code and commands and file and folder names in a `sans-serif font`, and on-screen components, such as menu names, in a **bold sans-serif font**. We use *italics for emphasis* and **bold occasionally for strong emphasis**.

Getting the Code Examples

You can download the `examples.zip` file containing the book's examples from Pearson's Companion Website for the book at:

```
https://www.pearsonglobaleditions.com
```

When the download completes, locate it on your system, and extract its `examples` folder into your user account's `Documents` folder:

- Windows: `C:\Users\`*YourAccount*`\Documents\examples`
- macOS or Linux: `~/Documents/examples`

Most operating systems have a built-in extraction tool. You also may use an archive tool such as 7-Zip (`www.7-zip.org`) or WinZip (`www.winzip.com`).

Structure of the `examples` Folder

You'll execute three kinds of examples in this book:

- Individual code snippets in the IPython interactive environment.
- Complete applications, which are known as scripts.
- Jupyter Notebooks—a convenient interactive, web-browser-based environment in which you can write and execute code and intermix the code with text, images and video.

We demonstrate each in Section 1.10's test drives.

The `examples` folder contains one subfolder per chapter. These are named `ch##`, where `##` is the two-digit chapter number `01` to `17`—for example, `ch01`. Except for Chapters 14, 16 and 17, each chapter's folder contains the following items:

- `snippets_ipynb`—A folder containing the chapter's Jupyter Notebook files.
- `snippets_py`—A folder containing Python source code files in which each code snippet we present is separated from the next by a blank line. You can copy and paste these snippets into IPython or into new Jupyter Notebooks that you create.
- Script files and their supporting files.

Chapter 14 contains one application. Chapters 16 and 17 explain where to find the files you need in the `ch16` and `ch17` folders, respectively.

Installing Anaconda

We use the easy-to-install Anaconda Python distribution with this book. It comes with almost everything you'll need to work with our examples, including:

- the IPython interpreter,
- most of the Python and data science libraries we use,
- a local Jupyter Notebooks server so you can load and execute our notebooks, and
- various other software packages, such as the Spyder Integrated Development Environment (IDE)—we use only IPython and Jupyter Notebooks in this book.

Download the Python 3.x Anaconda installer for Windows, macOS or Linux from:

```
https://www.anaconda.com/download/
```

When the download completes, run the installer and follow the on-screen instructions. To ensure that Anaconda runs correctly, do not move its files after you install it.

Updating Anaconda

Next, ensure that Anaconda is up to date. Open a command-line window on your system as follows:

- On macOS, open a **Terminal** from the **Applications** folder's **Utilities** subfolder.
- On Windows, open the **Anaconda Prompt** from the start menu. When doing this to update Anaconda (as you'll do here) or to install new packages (discussed momentarily), execute the **Anaconda Prompt** as an *administrator* by right-clicking, then selecting **More > Run as administrator**. (If you cannot find the Anaconda Prompt in the start menu, simply search for it in the **Type here to search** field at the bottom of your screen.)
- On Linux, open your system's **Terminal** or shell (this varies by Linux distribution).

In your system's command-line window, execute the following commands to update Anaconda's installed packages to their latest versions:

1. `conda update conda`
2. `conda update --all`

Package Managers

The `conda` command used above invokes the **conda package manager**—one of the two key Python package managers you'll use in this book. The other is **`pip`**. Packages contain the files required to install a given Python library or tool. Throughout the book, you'll use `conda` to install additional packages, unless those packages are not available through `conda`, in which case you'll use `pip`. Some people prefer to use `pip` exclusively as it currently supports more packages. If you ever have trouble installing a package with `conda`, try `pip` instead.

Installing the Prospector Static Code Analysis Tool

In the book's exercises, we ask you to analyze Python code using the Prospector analysis tool, which checks your Python code for common errors and helps you improve your code. To install Prospector and the Python libraries it uses, run the following command in the command-line window:

```
pip install prospector
```

Installing jupyter-matplotlib

We implement several animations using a visualization library called Matplotlib. To use them in Jupyter Notebooks, you must install a tool called `ipympl`. In the **Terminal**, **Anaconda Command Prompt** or shell you opened previously, execute the following commands[1] one at a time:

```
conda install -c conda-forge ipympl
conda install nodejs
jupyter labextension install @jupyter-widgets/jupyterlab-manager
jupyter labextension install jupyter-matplotlib
```

Installing the Other Packages

Anaconda comes with approximately 300 popular Python and data science packages for you, such as NumPy, Matplotlib, pandas, Regex, BeautifulSoup, requests, Bokeh, SciPy, SciKit-Learn, Seaborn, Spacy, sqlite, statsmodels and many more. The number of additional packages you'll need to install throughout the book will be small and we'll provide installation instructions as necessary. As you discover new packages, their documentation will explain how to install them.

Get a Twitter Developer Account

If you intend to use our "Data Mining Twitter" chapter and any Twitter-based examples in subsequent chapters, apply for a Twitter developer account. Twitter now requires registration for access to their APIs. To apply, fill out and submit the application at

```
https://developer.twitter.com/en/apply-for-access
```

Twitter reviews every application. At the time of this writing, personal developer accounts were being approved immediately and company-account applications were taking from several days to several weeks. Approval is not guaranteed.

Internet Connection Required in Some Chapters

While using this book, you'll need an Internet connection to install various additional Python libraries. In some chapters, you'll register for accounts with cloud-based services, mostly to use their free tiers. Some services require credit cards to verify your identity. In a few cases, you'll use services that are not free. In these cases, you'll take advantage of monetary credits provided by the vendors so you can try their services without incurring charges. **Caution: Some cloud-based services incur costs once you set them up. When**

1. `https://github.com/matplotlib/jupyter-matplotlib`.

you complete our case studies using such services, be sure to promptly delete the resources you allocated.

Slight Differences in Program Outputs

When you execute our examples, you might notice some differences between the results we show and your own results:

- Due to differences in how calculations are performed with floating-point numbers (like –123.45, 7.5 or 0.0236937) across operating systems, you might see minor variations in outputs—especially in digits far to the right of the decimal point.
- When we show outputs that appear in separate windows, we crop the windows to remove their borders.

Getting Your Questions Answered

Online forums enable you to interact with other Python programmers and get your Python questions answered. Popular Python and general programming forums include:

- `python-forum.io`
- `StackOverflow.com`
- `https://www.dreamincode.net/forums/forum/29-python/`

Also, many vendors provide forums for their tools and libraries. Most of the libraries you'll use in this book are managed and maintained at `github.com`. Some library maintainers provide support through the **Issues** tab on a given library's GitHub page. If you cannot find an answer to your questions online, please see our web page for the book at

`http://www.deitel.com`[1]

You're now ready to begin reading *Intro to Python for Computer Science and Data Sciences: Learning to Program with AI, Big Data and the Cloud*. We hope you enjoy the book!

1. Our website is undergoing a major upgrade. If you do not find something you need, please write to us directly at `deitel@deitel.com`.

Introduction to Computers and Python

1

Objectives

In this chapter you'll:

- Learn about exciting recent developments in computing.
- Learn computer hardware, software and Internet basics.
- Understand the data hierarchy from bits to databases.
- Understand the different types of programming languages.
- Understand object-oriented programming basics.
- Understand the strengths of Python and other leading programming languages.
- Understand the importance of libraries.
- Be introduced to key Python and data-science libraries you'll use in this book.
- Test-drive the IPython interpreter's interactive mode for executing Python code.
- Execute a Python script that animates a bar chart.
- Create and test-drive a web-browser-based Jupyter Notebook for executing Python code.
- Learn how big "big data" is and how quickly it's getting even bigger.
- Read a big-data case study on a mobile navigation app.
- Be introduced to artificial intelligence—at the intersection of computer science and data science.

1.1 Introduction
1.2 Hardware and Software
1.2.1 Moore's Law
1.2.2 Computer Organization
1.3 Data Hierarchy
1.4 Machine Languages, Assembly Languages and High-Level Languages
1.5 Introduction to Object Technology
1.6 Operating Systems
1.7 Python
1.8 It's the Libraries!
1.8.1 Python Standard Library
1.8.2 Data-Science Libraries
1.9 Other Popular Programming Languages
1.10 Test-Drives: Using IPython and Jupyter Notebooks
1.10.1 Using IPython Interactive Mode as a Calculator
1.10.2 Executing a Python Program Using the IPython Interpreter
1.10.3 Writing and Executing Code in a Jupyter Notebook
1.11 Internet and World Wide Web
1.11.1 Internet: A Network of Networks
1.11.2 World Wide Web: Making the Internet User-Friendly
1.11.3 The Cloud
1.11.4 Internet of Things
1.12 Software Technologies
1.13 How Big Is Big Data?
1.13.1 Big Data Analytics
1.13.2 Data Science and Big Data Are Making a Difference: Use Cases
1.14 Case Study—A Big-Data Mobile Application
1.15 Intro to Data Science: Artificial Intelligence—at the Intersection of CS and Data Science
Exercises

1.1 Introduction

Welcome to Python—one of the world's most widely used computer programming languages and, according to the *Popularity of Programming Languages (PYPL) Index*, the world's most popular.[1] You're probably familiar with many of the powerful tasks computers perform. In this textbook, you'll get intensive, hands-on experience writing Python instructions that command computers to perform those and other tasks. **Software** (that is, the Python instructions you write, which are also called **code**) controls **hardware** (that is, computers and related devices).

Here, we introduce terminology and concepts that lay the groundwork for the Python programming you'll learn in Chapters 2–11 and the big-data, artificial-intelligence and cloud-based case studies we present in Chapters 12–17. We'll introduce hardware and software concepts and overview the data hierarchy—from individual bits to databases, which store the massive amounts of data companies need to implement contemporary applications such as Google Search, Waze, Uber, Airbnb and a myriad of others.

We'll discuss the types of programming languages and introduce *object-oriented programming* terminology and concepts. You'll learn why Python has become so popular. We'll introduce the Python Standard Library and various data-science libraries that help you avoid "reinventing the wheel." You'll use these libraries to create *software objects* that you'll interact with to perform significant tasks with modest numbers of instructions. We'll introduce additional software technologies that you're likely to use as you develop software.

Next, you'll work through three test-drives showing how to execute Python code:

- In the first, you'll use IPython to execute Python instructions interactively and immediately see their results.

1. `https://pypl.github.io/PYPL.html` (as of January 2019).

- In the second, you'll execute a substantial Python application that will display an animated bar chart summarizing rolls of a six-sided die as they occur. You'll see the "Law of Large Numbers" in action. In Chapter 6, you'll build this application with the Matplotlib visualization library.
- In the last, we'll introduce Jupyter Notebooks using JupyterLab—an interactive, web-browser-based tool in which you can conveniently write and execute Python instructions. Jupyter Notebooks enable you to include text, images, audios, videos, animations and code.

In the past, most computer applications ran on "standalone" computers (that is, not networked together). Today's applications can be written with the aim of communicating among the world's computers via the Internet. We'll introduce the Internet, the World Wide Web, the Cloud and the Internet of Things (IoT), laying the groundwork for the contemporary applications you'll develop in Chapters 12–17.

You'll learn just how big "big data" is and how quickly it's getting even bigger. Next, we'll present a big-data case study on the Waze mobile navigation app, which uses many current technologies to provide dynamic driving directions that get you to your destination as quickly and as safely as possible. As we walk through those technologies, we'll mention where you'll use many of them in this book. The chapter closes with our first Intro to Data Science section in which we discuss a key intersection between computer science and data science—artificial intelligence.

1.2 Hardware and Software

Computers can perform calculations and make logical decisions phenomenally faster than human beings can. Many of today's personal computers can perform billions of calculations in one second—more than a human can perform in a lifetime. *Supercomputers* are already performing *thousands of trillions (quadrillions)* of instructions per second! IBM has developed the IBM Summit supercomputer, which can perform over 122 quadrillion calculations per second (122 *petaflops*)![2] To put that in perspective, *the IBM Summit supercomputer can perform in one second almost 16 million calculations for every person on the planet!*[3] And supercomputing upper limits are growing quickly.

Computers process data under the control of sequences of instructions called **computer programs** (or simply **programs**). These software programs guide the computer through ordered actions specified by people called computer **programmers**.

A computer consists of various physical devices referred to as hardware (such as the keyboard, screen, mouse, solid-state disks, hard disks, memory, DVD drives and processing units). Computing costs are *dropping dramatically*, due to rapid developments in hardware and software technologies. Computers that might have filled large rooms and cost millions of dollars decades ago are now inscribed on computer chips smaller than a fingernail, costing perhaps a few dollars each. Ironically, silicon is one of the most abundant materials on Earth—it's an ingredient in common sand. Silicon-chip technology has made computing so economical that computers have become a commodity.

2. `https://en.wikipedia.org/wiki/FLOPS`.
3. For perspective on how far computing performance has come, consider this: In his early computing days, Harvey Deitel used the Digital Equipment Corporation PDP-1 (`https://en.wikipedia.org/wiki/PDP-1`), which was capable of performing only 93,458 operations per second.

1.2.1 Moore's Law

Every year, you probably expect to pay at least a little more for most products and services. The opposite has been the case in the computer and communications fields, especially with regard to the hardware supporting these technologies. For many decades, hardware costs have fallen rapidly.

Every year or two, the capacities of computers have approximately *doubled* inexpensively. This remarkable trend often is called **Moore's Law**, named for the person who identified it in the 1960s, Gordon Moore, co-founder of Intel—one of the leading manufacturers of the processors in today's computers and embedded systems. Moore's Law *and related observations* apply especially to

- the amount of memory that computers have for programs,
- the amount of secondary storage (such as solid-state drive storage) they have to hold programs and data over longer periods of time, and
- their processor speeds—the speeds at which they *execute* their programs (that is, do their work).

Similar growth has occurred in the communications field—costs have plummeted as enormous demand for communications *bandwidth* (that is, information-carrying capacity) has attracted intense competition. We know of no other fields in which technology improves so quickly and costs fall so rapidly. Such phenomenal improvement is truly fostering the *Information Revolution*.

1.2.2 Computer Organization

Regardless of differences in *physical* appearance, computers can be envisioned as divided into various **logical units** or sections:

Input Unit

This "receiving" section obtains information (data and computer programs) from **input devices** and places it at the disposal of the other units for processing. Most user input is entered into computers through keyboards, touch screens and mouse devices. Other forms of input include receiving voice commands, scanning images and barcodes, reading from secondary storage devices (such as hard drives, Blu-ray Disc™ drives and USB flash drives—also called "thumb drives" or "memory sticks"), receiving video from a webcam and having your computer receive information from the Internet (such as when you stream videos from YouTube® or download e-books from Amazon). Newer forms of input include position data from a GPS device, motion and orientation information from an *accelerometer* (a device that responds to up/down, left/right and forward/backward acceleration) in a smartphone or wireless game controller (such as those for Microsoft® Xbox®, Nintendo Switch™ and Sony® PlayStation®) and voice input from intelligent assistants like Apple Siri®, Amazon Echo® and Google Home®.

Output Unit

This "shipping" section takes information the computer has processed and places it on various **output devices** to make it available for use outside the computer. Most information that's output from computers today is displayed on screens (including touch screens), printed on paper ("going green" discourages this), played as audio or video on smartphones, tablets, PCs and giant screens in sports stadiums, transmitted over the Internet or used to control other devices, such as self-driving cars, robots and "intelligent" appliances. Information is also commonly output to secondary storage devices, such as solid-state drives (SSDs), hard drives, DVD drives and USB flash drives. Popular recent forms of output are smartphone and game-controller vibration, virtual reality devices like Oculus Rift®, Sony® PlayStation® VR and Google Daydream View™ and Samsung Gear VR®, and mixed reality devices like Magic Leap® One and Microsoft HoloLens™.

Memory Unit

This rapid-access, relatively low-capacity "warehouse" section retains information that has been entered through the input unit, making it immediately available for processing when needed. The memory unit also retains processed information until it can be placed on output devices by the output unit. Information in the memory unit is *volatile*—it's typically lost when the computer's power is turned off. The memory unit is often called either **memory**, **primary memory** or **RAM** (Random Access Memory). Main memories on desktop and notebook computers contain as much as 128 GB of RAM, though 8 to 16 GB is most common. GB stands for gigabytes; a gigabyte is approximately one billion bytes. A **byte** is eight bits. A bit is either a `0` or a `1`.

Arithmetic and Logic Unit (ALU)

This "manufacturing" section performs *calculations*, such as addition, subtraction, multiplication and division. It also contains the *decision* mechanisms that allow the computer, for example, to compare two items from the memory unit to determine whether they're equal. In today's systems, the ALU is part of the next logical unit, the CPU.

Central Processing Unit (CPU)

This "administrative" section coordinates and supervises the operation of the other sections. The CPU tells the input unit when information should be read into the memory unit, tells the ALU when information from the memory unit should be used in calculations and tells the output unit when to send information from the memory unit to specific output devices. Most computers have **multicore processors** that implement multiple processors on a single integrated-circuit chip. Such processors can perform many operations simultaneously. A *dual-core processor* has two CPUs, a *quad-core processor* has four and an *octa-core processor* has eight. Intel has some processors with up to 72 cores. Today's desktop computers have processors that can execute billions of instructions per second.

Secondary Storage Unit

This is the long-term, high-capacity "warehousing" section. Programs or data not actively being used by the other units normally are placed on secondary storage devices (e.g., your *hard drive*) until they're again needed, possibly hours, days, months or even years later. Information on secondary storage devices is *persistent*—it's preserved even when the computer's power is turned off. Secondary storage information takes much longer to access than information in primary memory, but its cost per unit is much less. Examples of secondary storage devices include solid-state drives (SSDs), hard drives, read/write Blu-ray drives and USB flash drives. Many current drives hold terabytes (TB) of data—a **terabyte** is approximately one trillion bytes). Typical hard drives on desktop and notebook computers hold up to 4 TB, and some recent desktop-computer hard drives hold up to 15 TB.[4]

✓ Self Check for Section 1.2

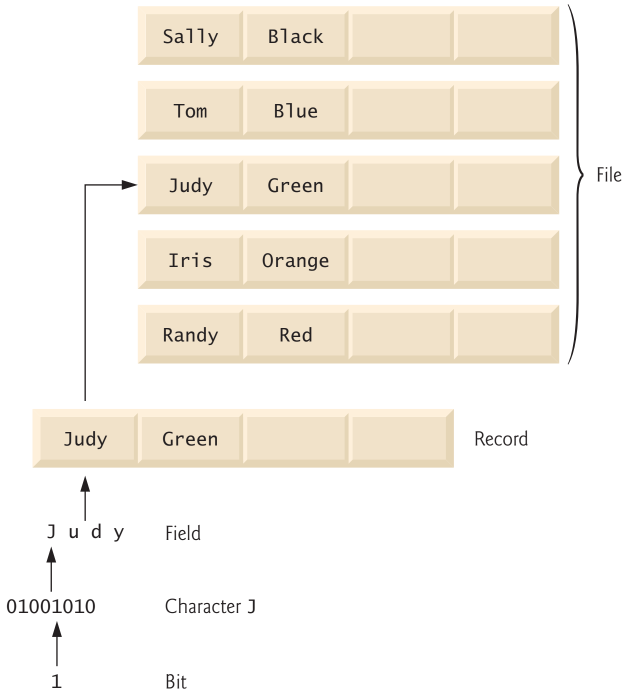
1 *(Fill-In)* Every year or two, the capacities of computers have approximately doubled inexpensively. This remarkable trend often is called ________.
Answer: Moore's Law.

2 *(True/False)* Information in the memory unit is *persistent*—it's preserved even when the computer's power is turned off
Answer: False. Information in the memory unit is *volatile*—it's typically lost when the computer's power is turned off.

3 *(Fill-In)* Most computers have ________ processors that implement multiple processors on a single integrated-circuit chip. Such processors can perform many operations simultaneously.
Answer: multicore.

1.3 Data Hierarchy

Data items processed by computers form a **data hierarchy** that becomes larger and more complex in structure as we progress from the simplest data items (called "bits") to richer ones, such as characters and fields. The following diagram illustrates a portion of the data hierarchy:



4. `https://www.zdnet.com/article/worlds-biggest-hard-drive-meet-western-digitals-15tb-monster/`.

Bits

A **bit** (short for "*b*inary dig*it*"—a digit that can assume one of *two* values) is the smallest data item in a computer. It can have the value `0` or `1`. Remarkably, the impressive functions performed by computers involve only the simplest manipulations of `0`s and `1`s—*examining a bit's value*, *setting a bit's value* and *reversing a bit's value* (from `1` to `0` or from `0` to `1`). Bits for the basis of the binary number system, which you can study in-depth in our online "Number Systems" appendix.

Characters

Work with data in the low-level form of bits is tedious. Instead, people prefer to work with *decimal digits* (`0`–`9`), *letters* (`A`–`Z` and `a`–`z`) and *special symbols* such as

```
$ @ % & * ( ) – + " : ; , ? /
```

Digits, letters and special symbols are known as **characters**. The computer's **character set** contains the characters used to write programs and represent data items. Computers process only `1`s and `0`s, so a computer's character set represents every character as a pattern of `1`s and `0`s. Python uses **Unicode**® characters that are composed of one, two, three or four bytes (8, 16, 24 or 32 bits, respectively)—known as **UTF-8 encoding**.[5]

Unicode contains characters for many of the world's languages. The **ASCII (American Standard Code for Information Interchange)** character set is a subset of Unicode that represents letters (a–z and A–Z), digits and some common special characters. You can view the ASCII subset of Unicode at

```
https://www.unicode.org/charts/PDF/U0000.pdf
```

The Unicode charts for all languages, symbols, emojis and more are viewable at

```
http://www.unicode.org/charts/
```

Fields

Just as characters are composed of bits, **fields** are composed of characters or bytes. A field is a group of characters or bytes that conveys meaning. For example, a field consisting of uppercase and lowercase letters can be used to represent a person's name, and a field consisting of decimal digits could represent a person's age.

Records

Several related fields can be used to compose a **record**. In a payroll system, for example, the record for an employee might consist of the following fields (possible types for these fields are shown in parentheses):

- Employee identification number (a whole number).
- Name (a string of characters).
- Address (a string of characters).
- Hourly pay rate (a number with a decimal point).
- Year-to-date earnings (a number with a decimal point).
- Amount of taxes withheld (a number with a decimal point).

5. `https://docs.python.org/3/howto/unicode.html`.

Thus, a record is a group of related fields. All the fields listed above belong to the same employee. A company might have many employees and a payroll record for each.

Files

A **file** is a group of related records. More generally, a file contains arbitrary data in arbitrary formats. In some operating systems, a file is viewed simply as a *sequence of bytes*—any organization of the bytes in a file, such as organizing the data into records, is a view created by the application programmer. You'll see how to do that in Chapter 9, "Files and Exceptions." It's not unusual for an organization to have many files, some containing billions, or even trillions, of characters of information.

Databases

A **database** is a collection of data organized for easy access and manipulation. The most popular model is the *relational database*, in which data is stored in simple *tables*. A table includes *records* and *fields*. For example, a table of students might include first name, last name, major, year, student ID number and grade-point-average fields. The data for each student is a record, and the individual pieces of information in each record are the fields. You can *search*, *sort* and otherwise manipulate the data, based on its relationship to multiple tables or databases. For example, a university might use data from the student database in combination with data from databases of courses, on-campus housing, meal plans, etc. We discuss databases in Chapter 17, "Big Data: Hadoop, Spark, NoSQL and IoT."

Big Data

The table below shows some common byte measurements:

Unit	Bytes	Which is approximately
1 kilobyte (KB)	1024 bytes	10^3 (1024) bytes exactly
1 megabyte (MB)	1024 kilobytes	10^6 (1,000,000) bytes
1 gigabyte (GB)	1024 megabytes	10^9 (1,000,000,000) bytes
1 terabyte (TB)	1024 gigabytes	10^{12} (1,000,000,000,000) bytes
1 petabyte (PB)	1024 terabytes	10^{15} (1,000,000,000,000,000) bytes
1 exabyte (EB)	1024 petabytes	10^{18} (1,000,000,000,000,000,000) bytes
1 zettabyte (ZB)	1024 exabytes	10^{21} (1,000,000,000,000,000,000,000) bytes

The amount of data being produced worldwide is enormous and its growth is accelerating. **Big data** applications deal with massive amounts of data. This field is growing quickly, creating lots of opportunity for software developers. Millions of IT jobs globally already are supporting big data applications. Section 1.13 discusses big data in more depth. You'll study big data and associated technologies in Chapter 17.

✓ Self Check

1 *(Fill-In)* A(n) ________ (short for "binary digit"—a digit that can assume one of two values) is the smallest data item in a computer.
Answer: bit.

2 *(True/False)* In some operating systems, a file is viewed simply as a sequence of bytes—any organization of the bytes in a file, such as organizing the data into records, is a view created by the application programmer.
Answer: True.

3 *(Fill-In)* A database is a collection of data organized for easy access and manipulation. The most popular model is the ________ database, in which data is stored in simple tables.
Answer: *relational*

1.4 Machine Languages, Assembly Languages and High-Level Languages

Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate *translation* steps. Hundreds of such languages are in use today. These may be divided into three general types:

1. Machine languages.
2. Assembly languages.
3. High-level languages.

Machine Languages

Any computer can directly understand only its own **machine language**, defined by its hardware design. Machine languages generally consist of strings of numbers (ultimately reduced to 1s and 0s) that instruct computers to perform their most elementary operations one at a time. Machine languages are *machine dependent* (a particular machine language can be used on only one type of computer). Such languages are cumbersome for humans. For example, here's a section of an early machine-language payroll program that adds overtime pay to base pay and stores the result in gross pay:

```
+1300042774
+1400593419
+1200274027
```

Assembly Languages and Assemblers

Programming in machine language was simply too slow and tedious for most programmers. Instead of using the strings of numbers that computers could directly understand, programmers began using English-like abbreviations to represent elementary operations. These abbreviations formed the basis of **assembly languages**. *Translator programs* called **assemblers** were developed to convert assembly-language programs to machine language at computer speeds. The following section of an assembly-language payroll program also adds overtime pay to base pay and stores the result in gross pay:

```
load    basepay
add     overpay
store   grosspay
```

Although such code is clearer to humans, it's incomprehensible to computers until translated to machine language.

High-Level Languages and Compilers

With the advent of assembly languages, computer usage increased rapidly, but programmers still had to use numerous instructions to accomplish even the simplest tasks. To speed the programming process, **high-level languages** were developed in which single statements could be written to accomplish substantial tasks. A typical high-level-language program contains many statements, known as the program's **source code**.

Translator programs called **compilers** convert high-level-language source code into machine language. High-level languages allow you to write instructions that look almost like everyday English and contain commonly used mathematical notations. A payroll program written in a high-level language might contain a *single* statement such as

```
grossPay = basePay + overTimePay
```

From the programmer's standpoint, high-level languages are preferable to machine and assembly languages. Python is among the world's most widely used high-level programming languages.

Interpreters

Compiling a large high-level language program into machine language can take considerable computer time. *Interpreter* programs, developed to execute high-level language programs directly, avoid the delay of compilation, although they run slower than compiled programs. The most widely used Python implementation—CPython (which is written in the C programming language)—uses a clever mixture of compilation and interpretation to run programs.[6]

✓ Self Check

1 *(Fill-In)* Translator programs called ________ convert assembly-language programs to machine language at computer speeds.
Answer: assemblers.

2 *(Fill-In)* ________ programs, developed to execute high-level-language programs directly, avoid the delay of compilation, although they run slower than compiled programs
Answer: Interpreter.

3 *(True/False)* High-level languages allow you to write instructions that look almost like everyday English and contain commonly used mathematical notations.
Answer: True.

1.5 Introduction to Object Technology

As demands for new and more powerful software are soaring, building software quickly, correctly and economically is important. *Objects*, or more precisely, the *classes* objects come from, are essentially *reusable* software components. There are date objects, time objects, audio objects, video objects, automobile objects, people objects, etc. Almost any *noun* can be reasonably represented as a software object in terms of *attributes* (e.g., name, color and size) and *behaviors* (e.g., calculating, moving and communicating). Software-development groups can use a modular, object-oriented design-and-implementation approach to be

6. `https://opensource.com/article/18/4/introduction-python-bytecode`.

much more productive than with earlier popular techniques like "structured programming." Object-oriented programs are often easier to understand, correct and modify.

Automobile as an Object

To help you understand objects and their contents, let's begin with a simple analogy. Suppose you want to *drive a car and make it go faster by pressing its accelerator pedal.* What must happen before you can do this? Well, before you can drive a car, someone has to *design* it. A car typically begins as engineering drawings, similar to the *blueprints* that describe the design of a house. These drawings include the design for an accelerator pedal. The pedal *hides* from the driver the complex mechanisms that make the car go faster, just as the brake pedal "hides" the mechanisms that slow the car, and the steering wheel "hides" the mechanisms that turn the car. This enables people with little or no knowledge of how engines, braking and steering mechanisms work to drive a car easily.

Just as you cannot cook meals in the blueprint of a kitchen, you cannot drive a car's engineering drawings. Before you can drive a car, it must be *built* from the engineering drawings that describe it. A completed car has an *actual* accelerator pedal to make it go faster, but even that's not enough—the car won't accelerate on its own (hopefully!), so the driver must *press* the pedal to accelerate the car.

Methods and Classes

Let's use our car example to introduce some key object-oriented programming concepts. Performing a task in a program requires a **method**. The method houses the program statements that perform its tasks. The method hides these statements from its user, just as the accelerator pedal of a car hides from the driver the mechanisms of making the car go faster. In Python, a program unit called a **class** houses the set of methods that perform the class's tasks. For example, a class that represents a bank account might contain one method to *deposit* money to an account, another to *withdraw* money from an account and a third to *inquire* what the account's balance is. A class is similar in concept to a car's engineering drawings, which house the design of an accelerator pedal, steering wheel, and so on.

Instantiation

Just as someone has to *build a car* from its engineering drawings before you can drive a car, you must *build an object* of a class before a program can perform the tasks that the class's methods define. The process of doing this is called *instantiation*. An object is then referred to as an **instance** of its class.

Reuse

Just as a car's engineering drawings can be *reused* many times to build many cars, you can *reuse* a class many times to build many objects. Reuse of existing classes when building new classes and programs saves time and effort. Reuse also helps you build more reliable and effective systems because existing classes and components often have undergone extensive *testing*, *debugging* (that is, finding and removing errors) and *performance tuning*. Just as the notion of *interchangeable parts* was crucial to the Industrial Revolution, reusable classes are crucial to the software revolution that has been spurred by object technology.

In Python, you'll typically use a *building-block approach* to create your programs. To avoid reinventing the wheel, you'll use existing high-quality pieces wherever possible. This software reuse is a key benefit of object-oriented programming.

Messages and Method Calls

When you drive a car, pressing its gas pedal sends a *message* to the car to perform a task—that is, to go faster. Similarly, you *send messages to an object*. Each message is implemented as a **method call** that tells a method of the object to perform its task. For example, a program might call a bank-account object's *deposit* method to increase the account's balance.

Attributes and Instance Variables

A car, besides having capabilities to accomplish tasks, also has *attributes*, such as its color, its number of doors, the amount of gas in its tank, its current speed and its record of total miles driven (i.e., its odometer reading). Like its capabilities, the car's attributes are represented as part of its design in its engineering diagrams (which, for example, include an odometer and a fuel gauge). As you drive an actual car, these attributes are carried along with the car. Every car maintains its *own* attributes. For example, each car knows how much gas is in its own gas tank, but *not* how much is in the tanks of *other* cars.

An object, similarly, has attributes that it carries along as it's used in a program. These attributes are specified as part of the object's class. For example, a bank-account object has a *balance attribute* that represents the amount of money in the account. Each bank-account object knows the balance in the account it represents, but *not* the balances of the *other* accounts in the bank. Attributes are specified by the class's **instance variables**. A class's (and its object's) attributes and methods are intimately related, so classes wrap together their attributes and methods.

Inheritance

A new class of objects can be created conveniently by **inheritance**—the new class (called the **subclass**) starts with the characteristics of an existing class (called the **superclass**), possibly customizing them and adding unique characteristics of its own. In our car analogy, an object of class "convertible" certainly *is an* object of the more *general* class "automobile," but more *specifically*, the roof can be raised or lowered.

Object-Oriented Analysis and Design (OOAD)

Soon you'll be writing programs in Python. How will you create the **code** (i.e., the program instructions) for your programs? Perhaps, like many programmers, you'll simply turn on your computer and start typing. This approach may work for small programs (like the ones we present in the early chapters of the book), but what if you were asked to create a software system to control thousands of automated teller machines for a major bank? Or suppose you were asked to work on a team of 1,000 software developers building the next generation of the U.S. air traffic control system? For projects so large and complex, you should not simply sit down and start writing programs.

To create the best solutions, you should follow a detailed **analysis** process for determining your project's **requirements** (i.e., defining *what* the system is supposed to do), then develop a **design** that satisfies them (i.e., specifying *how* the system should do it). Ideally, you'd go through this process and carefully review the design (and have your design reviewed by other software professionals) before writing any code. If this process involves analyzing and designing your system from an object-oriented point of view, it's called an **object-oriented analysis-and-design (OOAD) process**. Languages like Python are object-oriented. Programming in such a language, called **object-oriented programming (OOP)**, allows you to implement an object-oriented design as a working system.

✓ Self Check for Section 1.5

1 *(Fill-In)* To create the best solutions, you should follow a detailed analysis process for determining your project's ________ (i.e., defining *what* the system is supposed to do) and developing a design that satisfies them (i.e., specifying *how* the system should do it).
Answer: requirements.

2 *(Fill-In)* The size, shape, color and weight of an object are ________ of the object's class.
Answer: attributes.

3 *(True/False)* Objects, or more precisely, the classes objects come from, are essentially reusable software components.
Answer: True.

1.6 Operating Systems

Operating systems are software systems that make using computers more convenient for users, application developers and system administrators. They provide services that allow each application to execute safely, efficiently and *concurrently* with other applications. The software that contains the core components of the operating system is called the **kernel**. Linux, Windows and macOS are popular desktop computer operating systems—you can use any of these with this book. The most popular mobile operating systems used in smartphones and tablets are Google's Android and Apple's iOS.

Windows—A Proprietary Operating System

In the mid-1980s, Microsoft developed the **Windows operating system**, consisting of a graphical user interface built on top of DOS (Disk Operating System)—an enormously popular personal-computer operating system that users interacted with by typing commands. Windows 10 is Microsoft's latest operating system—it includes the Cortana personal assistant for voice interactions. Windows is a *proprietary* operating system—it's controlled by Microsoft exclusively. Windows is by far the world's most widely used desktop operating system.

Linux—An Open-Source Operating System

The **Linux operating system** is among the greatest successes of the *open-source* movement. **Open-source software** departs from the *proprietary* software development style that dominated software's early years. With open-source development, individuals and companies *contribute* their efforts in developing, maintaining and evolving software in exchange for the right to use that software for their own purposes, typically at *no charge*. Open-source code is often scrutinized by a much larger audience than proprietary software, so errors often get removed faster. Open source also encourages innovation.

There are many organizations in the open-source community. Some key ones are:

- **Python Software Foundation** (responsible for Python).
- **GitHub** (provides tools for managing open-source projects—it has millions of them under development).
- The **Apache Software Foundation** (originally the creators of the Apache web server, they now oversee 350 open-source projects, including several big data infrastructure technologies we present in Chapter 17.

- The **Eclipse Foundation** (the Eclipse Integrated Development Environment helps programmers conveniently develop software)
- The **Mozilla Foundation** (creators of the Firefox web browser)
- **OpenML** (which focuses on open-source tools and data for machine learning—you'll explore machine learning in Chapter 15).
- **OpenAI** (which does research on artificial intelligence and publishes open-source tools used in AI reinforcement-learning research).
- **OpenCV** (which focuses on open-source computer-vision tools that can be used across a range of operating systems and programming languages—you'll study computer-vision applications in Chapter 16).

Rapid improvements to computing and communications, decreasing costs and open-source software have made it much easier and more economical to create software-based businesses now than just a decade ago. A great example is Facebook, which was launched from a college dorm room and built with open-source software.

The **Linux kernel** is the core of the most popular open-source, freely distributed, full-featured operating system. It's developed by a loosely organized team of volunteers and is popular in servers, personal computers and embedded systems (such as the computer systems at the heart of smartphones, smart TVs and automobile systems). Unlike that of proprietary operating systems like Microsoft's Windows and Apple's macOS, Linux source code (the program code) is available to the public for examination and modification and is free to download and install. As a result, Linux users benefit from a huge community of developers actively debugging and improving the kernel, and the ability to customize the operating system to meet specific needs.

Apple's macOS and Apple's iOS for iPhone® and iPad® Devices

Apple, founded in 1976 by Steve Jobs and Steve Wozniak, quickly became a leader in personal computing. In 1979, Jobs and several Apple employees visited Xerox PARC (Palo Alto Research Center) to learn about Xerox's desktop computer that featured a graphical user interface (GUI). That GUI served as the inspiration for the Apple Macintosh, launched in 1984.

The Objective-C programming language, created by Stepstone in the early 1980s, added capabilities for object-oriented programming (OOP) to the C programming language. Steve Jobs left Apple in 1985 and founded NeXT Inc. In 1988, NeXT licensed Objective-C from Stepstone and developed an Objective-C compiler and libraries which were used as the platform for the NeXTSTEP operating system's user interface, and Interface Builder—used to construct graphical user interfaces.

Jobs returned to Apple in 1996 when they bought NeXT. Apple's **macOS operating system** is a descendant of NeXTSTEP. Apple's proprietary operating system, **iOS**, is derived from macOS and is used in the iPhone, iPad, Apple Watch and Apple TV devices. In 2014, Apple introduced its new Swift programming language, which became open source in 2015. The iOS app-development community has largely shifted from Objective-C to Swift.

Google's Android

Android—the fastest growing mobile and smartphone operating system—is based on the Linux kernel and the Java programming language. Android is open source and free.

According to idc.com, as of 2018, Android had 86.8% of the global smartphone market share, compared to 13.2% for Apple.[7] The Android operating system is used in numerous smartphones, e-reader devices, tablets, in-store touch-screen kiosks, cars, robots, multimedia players and more.

Billions of Devices

In use today are Billions of personal computers and an even larger number of mobile devices. The following table lists many computerized devices. The explosive growth of mobile phones, tablets and other devices is creating significant opportunities for programming mobile apps. There are now various tools that enable you to use Python for Android and iOS app development, including BeeWare, Kivy, PyMob, Pythonista and others. Many are **cross-platform**, meaning that you can use them to develop apps that will run portably on Android, iOS and other platforms (like the web).

Computerized devices		
Access control systems	Airplane systems	ATMs
Automobiles	Blu-ray Disc™ players	Building controls
Cable boxes	Copiers	Credit cards
CT scanners	Desktop computers	e-Readers
Game consoles	GPS navigation systems	Home appliances
Home security systems	Internet-of-Things gateways	Light switches
Logic controllers	Lottery systems	Medical devices
Mobile phones	MRIs	Network switches
Optical sensors	Parking meters	Personal computers
Point-of-sale terminals	Printers	Robots
Routers	Servers	Smartcards
Smart meters	Smartpens	Smartphones
Tablets	Televisions	Thermostats
Transportation passes	TV set-top boxes	Vehicle diagnostic systems

✓ Self Check for Section 1.6

1 *(Fill-In)* Windows is a(n) _______ operating system—it's controlled by Microsoft exclusively.
Answer: proprietary.

2 *(True/False)* Proprietary code is often scrutinized by a much larger audience than open-source software, so errors often get removed faster.
Answer: False. Open-source code is often scrutinized by a much larger audience than proprietary software, so errors often get removed faster.

3 *(True/False)* iOS dominates the global smartphone market over Android.
Answer: False. Android currently controls 88% of the smartphone market.

7. `https://www.idc.com/promo/smartphone-market-share/os`.

1.7 Python

Python is an object-oriented scripting language that was released publicly in 1991. It was developed by Guido van Rossum of the National Research Institute for Mathematics and Computer Science in Amsterdam.

Python has rapidly become one of the world's most popular programming languages. It's now particularly popular for educational and scientific computing,[8] and it recently surpassed the programming language R as the most popular data-science programming language.[9,10,11] Here are some reasons why Python is popular and everyone should consider learning it:[12,13,14]

- It's open source, free and widely available with a massive open-source community.
- It's easier to learn than languages like C, C++, C# and Java, enabling novices and professional developers to get up to speed quickly.
- It's easier to read than many other popular programming languages.
- It's widely used in education.[15]
- It enhances developer productivity with extensive standard libraries and *thousands* of third-party open-source libraries, so programmers can write code faster and perform complex tasks with minimal code. We'll say more about this in Section 1.8.
- There are massive numbers of free open-source Python applications.
- It's popular in web development (e.g., Django, Flask).
- It supports popular programming paradigms—procedural, functional, object-oriented and reflective.[16] We'll begin introducing functional-style programming features in Chapter 4 and use them in subsequent chapters.
- It simplifies concurrent programming—with asyncio and async/await, you're able to write single-threaded concurrent code[17], greatly simplifying the inherently complex processes of writing, debugging and maintaining that code.[18]
- There are lots of capabilities for enhancing Python performance.
- It's used to build anything from simple scripts to complex apps with massive numbers of users, such as Dropbox, YouTube, Reddit, Instagram and Quora.[19]

8. `https://www.oreilly.com/ideas/5-things-to-watch-in-python-in-2017`.
9. `https://www.kdnuggets.com/2017/08/python-overtakes-r-leader-analytics-data-science.html`.
10. `https://www.r-bloggers.com/data-science-job-report-2017-r-passes-sas-but-python-leaves-them-both-behind/`.
11. `https://www.oreilly.com/ideas/5-things-to-watch-in-python-in-2017`.
12. `https://dbader.org/blog/why-learn-python`.
13. `https://simpleprogrammer.com/2017/01/18/7-reasons-why-you-should-learn-python/`.
14. `https://www.oreilly.com/ideas/5-things-to-watch-in-python-in-2017`.
15. Tollervey, N., *Python in Education: Teach, Learn, Program* (O'Reilly Media, Inc., 2015).
16. `https://en.wikipedia.org/wiki/Python_(programming_language)`.
17. `https://docs.python.org/3/library/asyncio.html`.
18. `https://www.oreilly.com/ideas/5-things-to-watch-in-python-in-2017`.
19. `https://www.hartmannsoftware.com/Blog/Articles_from_Software_Fans/Most-Famous-Software-Programs-Written-in-Python`.

- It's popular in artificial intelligence, which is enjoying explosive growth, in part because of its special relationship with data science.
- It's widely used in the financial community.[20]
- There's an extensive job market for Python programmers across many disciplines, especially in data-science-oriented positions, and Python jobs are among the highest paid of all programming jobs.[21,22]

Anaconda Python Distribution

We use the Anaconda Python distribution because it's easy to install on Windows, macOS and Linux and supports the latest versions of Python (3.7 at the time of this writing), the IPython interpreter (introduced in Section 1.10.1) and Jupyter Notebooks (introduced in Section 1.10.3). Anaconda also includes other software packages and libraries commonly used in Python programming and data science, allowing students to focus on learning Python, computer science and data science, rather than software installation issues. The IPython interpreter[23] has features that help students and professionals explore, discover and experiment with Python, the Python Standard Library and the extensive set of third-party libraries.

Zen of Python

We adhere to Tim Peters' *The Zen of Python*, which summarizes Python creator Guido van Rossum's design principles for the language. This list can be viewed in IPython with the command `import this`. The Zen of Python is defined in Python Enhancement Proposal (PEP) 20. "A PEP is a design document providing information to the Python community, or describing a new feature for Python or its processes or environment."[24]

✓ Self Check

1 *(Fill-In)* The ________ summarizes Python creator Guido van Rossum's design principles for the Python language.
Answer: Zen of Python.

2 *(True/False)* The Python language supports popular programming paradigms—procedural, functional, object-oriented and reflective.
Answer: True.

3 *(True/False)* R is most the popular data-science programming language.
Answer: False. Python recently surpassed R as the most popular data-science programming language.

20. Kolanovic, M. and R. Krishnamachari, *Big Data and AI Strategies: Machine Learning and Alternative Data Approach to Investing* (J.P. Morgan, 2017).
21. `https://www.infoworld.com/article/3170838/developer/get-paid-10-programming-languages-to-learn-in-2017.html`.
22. `https://medium.com/@ChallengeRocket/top-10-of-programming-languages-with-the-highest-salaries-in-2017-4390f468256e`.
23. `https://ipython.org/`.
24. `https://www.python.org/dev/peps/pep-0001/`.

1.8 It's the Libraries!

Throughout the book, we focus on using existing libraries to help you avoid "reinventing the wheel," thus leveraging your program-development efforts. Often, rather than developing lots of original code—a costly and time-consuming process—you can simply create an object of a pre-existing library class, which takes only a single Python statement. So, libraries will help you perform significant tasks with modest amounts of code. You'll use a broad range of Python standard libraries, data-science libraries and other third-party libraries.

1.8.1 Python Standard Library

The **Python Standard Library** provides rich capabilities for text/binary data processing, mathematics, functional-style programming, file/directory access, data persistence, data compression/archiving, cryptography, operating-system services, concurrent programming, interprocess communication, networking protocols, JSON/XML/other Internet data formats, multimedia, internationalization, GUI, debugging, profiling and more. The following table lists some of the Python Standard Library modules that we use in examples or that you'll explore in the exercises.

Some of the Python Standard Library modules we use in the book

- **`collections`**—Additional data structures beyond lists, tuples, dictionaries and sets.
- **`csv`**—Processing comma-separated value files.
- **`datetime`**, **`time`**—Date and time manipulations.
- **`decimal`**—Fixed-point and floating-point arithmetic, including monetary calculations.
- **`doctest`**—Simple unit testing via validation tests and expected results embedded in docstrings.
- **`json`**—JavaScript Object Notation (JSON) processing for use with web services and NoSQL document databases.
- **`math`**—Common math constants and operations.
- **`os`**—Interacting with the operating system.
- **`timeit`**—Performance analysis.
- **`queue`**—First-in, first-out data structure.
- **`random`**—Pseudorandom numbers.
- **`re`**—Regular expressions for pattern matching.
- **`sqlite3`**—SQLite relational database access.
- **`statistics`**—Mathematical statistics functions like `mean`, `median`, `mode` and `variance`.
- **`string`**—String processing.
- **`sys`**—Command-line argument processing; standard input, standard output and standard error streams.

1.8.2 Data-Science Libraries

Python has an enormous and rapidly growing community of open-source developers in many fields. One of the biggest reasons for Python's popularity is the extraordinary range of open-source libraries developed by the open-source community. One of our goals is to create examples, exercises, projects (EEPs) and implementation case studies that give you an engaging, challenging and entertaining introduction to Python programming, while also involving you in hands-on data science, key data-science libraries and more. You'll be amazed at the substantial tasks you can accomplish in just a few lines of code. The following table lists various popular data-science libraries. You'll use many of these as you work through our data-science examples, exercises and projects. For visualization, we'll focus primarily on Matplotlib and Seaborn, but there are many more. For a nice summary of Python visualization libraries see `http://pyviz.org/`.

Popular Python libraries used in data science

Scientific Computing and Statistics
NumPy (Numerical Python)—Python does not have a built-in array data structure. It uses lists, which are convenient but relatively slow. NumPy provides the more efficient `ndarray` data structure to represent lists and matrices, and it also provides routines for processing such data structures.

SciPy (Scientific Python)—Built on NumPy, SciPy adds routines for scientific processing, such as integrals, differential equations, additional matrix processing and more. `scipy.org` controls SciPy and NumPy.

StatsModels—Provides support for estimations of statistical models, statistical tests and statistical data exploration.

Data Manipulation and Analysis
Pandas—An extremely popular library for data manipulations. Pandas makes abundant use of NumPy's `ndarray`. Its two key data structures are `Series` (one dimensional) and `DataFrames` (two dimensional).

Visualization
Matplotlib—A highly customizable visualization and plotting library. Supported plots include regular, scatter, bar, contour, pie, quiver, grid, polar axis, 3D and text.

Seaborn—A higher-level visualization library built on Matplotlib. Seaborn adds a nicer look-and-feel, additional visualizations and enables you to create visualizations with less code.

Machine Learning, Deep Learning and Reinforcement Learning
scikit-learn—Top machine-learning library. Machine learning is a subset of AI. Deep learning is a subset of machine learning that focuses on neural networks.

Keras—One of the easiest to use deep-learning libraries. Keras runs on top of TensorFlow (Google), CNTK (Microsoft's cognitive toolkit for deep learning) or Theano (Université de Montréal).

TensorFlow—From Google, this is the most widely used deep learning library. TensorFlow works with GPUs (graphics processing units) or Google's custom TPUs (Tensor processing units) for performance. TensorFlow is important in AI and big data analytics—where processing demands are enormous. You'll use the version of Keras that's built into TensorFlow.

OpenAI Gym—A library and environment for developing, testing and comparing reinforcement-learning algorithms. You'll explore this in the Chapter 16 exercises.

Natural Language Processing (NLP)
NLTK (Natural Language Toolkit)—Used for natural language processing (NLP) tasks.

TextBlob—An object-oriented NLP text-processing library built on the NLTK and pattern NLP libraries. TextBlob simplifies many NLP tasks.

Gensim—Similar to NLTK. Commonly used to build an index for a collection of documents, then determine how similar another document is to each of those in the index. You'll explore this in the Chapter 12 exercises.

✓ Self Check for Section 1.8

1 *(Fill-In)* ________ help you avoid "reinventing the wheel," thus leveraging your program-development efforts.
Answer: Libraries.

2 *(Fill-In)* The ________ provides rich capabilities for many common Python programming tasks.
Answer: Python Standard Library.

1.9 Other Popular Programming Languages

The following is a brief introduction to several other popular programming languages—in the next section, we take a deeper look at Python:

- *Basic* was developed in the 1960s at Dartmouth College to familiarize novices with programming techniques. Many of its latest versions are object-oriented.
- *C* was developed in the early 1970s by Dennis Ritchie at Bell Laboratories. It initially became widely known as the UNIX operating system's development language. Today, most code for general-purpose operating systems and other performance-critical systems is written in C or C++.
- *C++*, which is based on C, was developed by Bjarne Stroustrup in the early 1980s at Bell Laboratories. C++ provides features that enhance the C language and adds capabilities for object-oriented programming.
- *Java*—Sun Microsystems in 1991 funded an internal corporate research project led by James Gosling, which resulted in the C++-based object-oriented programming language called Java. A key goal of Java is to enable developers to write programs that will run on a great variety of computer systems. This is called "write once, run anywhere." Java is used to develop enterprise applications, to enhance the functionality of web servers (the computers that provide the content to our web browsers), to provide applications for consumer devices (e.g., smartphones, tablets, television set-top boxes, appliances, automobiles and more) and for many other purposes. Java was originally the key language for developing Android smartphone and tablet apps, though several other languages are now supported.
- *C#* (based on C++ and Java) is one of Microsoft's three primary object-oriented programming languages—the other two are Visual C++ and Visual Basic. C# was developed to integrate the web into computer applications and is now widely used to develop many types of applications. As part of Microsoft's many open-source initiatives implemented over the last few years, they now offer open-source versions of C# and Visual Basic.
- *JavaScript* is the most widely used scripting language. It's primarily used to add programmability to web pages—for example, animations and interactivity with the user. All major web browsers support it. Many Python visualization libraries output JavaScript as part of visualizations that you can interact with in your web browser. Tools like NodeJS also enable JavaScript to run outside of web browsers.
- *Swift*, which was introduced in 2014, is Apple's programming language for developing iOS and macOS apps. Swift is a contemporary language that includes popular features from languages such as Objective-C, Java, C#, Ruby, Python and others. Swift is open source, so it can be used on non-Apple platforms as well.
- *R* is a popular open-source programming language for statistical applications and visualization. Python and R are the two most widely used data-science languages.

✓ **Self Check**

1 *(Fill-In)* Today, most code for general-purpose operating systems and other performance-critical systems is written in ________.
Answer: C or C++.

2 *(Fill-In)* A key goal of ________ is to enable developers to write programs that will run on a great variety of computer systems and computer-controlled devices. This is sometimes called "write once, run anywhere."
Answer: Java.

1.10 Test-Drives: Using IPython and Jupyter Notebooks

In this section, you'll test-drive the IPython interpreter[25] in two modes:

- In **interactive mode**, you'll enter small bits of Python code called **snippets** and immediately see their results.
- In **script mode**, you'll execute code loaded from a file that has the `.py` extension (short for Python). Such files are called **scripts** or **programs**, and they're generally longer than the code snippets you'll do in interactive mode.

Then, you'll learn how to use the browser-based environment known as the Jupyter Notebook for writing and executing Python code.[26]

1.10.1 Using IPython Interactive Mode as a Calculator

Let's use IPython interactive mode to evaluate simple arithmetic expressions.

Entering IPython in Interactive Mode

First, open a command-line window on your system:

- On macOS, open a **Terminal** from the **Applications** folder's **Utilities** subfolder.
- On Windows, open the **Anaconda Command Prompt** from the start menu.
- On Linux, open your system's **Terminal** or shell (this varies by Linux distribution).

In the command-line window, type `ipython`, then press *Enter* (or *Return*). You'll see text like the following, this varies by platform and by IPython version:

```
Python 3.7.0 | packaged by conda-forge | (default, Jan 20 2019, 17:24:52)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

The text "`In [1]:`" is a *prompt*, indicating that IPython is waiting for your input. You can type `?` for help or begin entering snippets, as you'll do momentarily.

25. Before reading this section, follow the instructions in the Before You Begin section to install the Anaconda Python distribution, which contains the IPython interpreter.
26. Jupyter supports many programming languages by installing their "kernels." For more information see `https://github.com/jupyter/jupyter/wiki/Jupyter-kernels`.

Evaluating Expressions

In interactive mode, you can evaluate expressions:

```
In [1]: 45 + 72
Out[1]: 117

In [2]:
```

After you type `45 + 72` and press *Enter*, IPython *reads* the snippet, *evaluates* it and *prints* its result in `Out[1]`.[27] Then IPython displays the `In [2]` prompt to show that it's waiting for you to enter your second snippet. For each new snippet, IPython adds 1 to the number in the square brackets. Each `In [1]` prompt in the book indicates that we've started a new interactive session. We generally do that for each new section of a chapter.

Let's evaluate a more complex expression:

```
In [2]: 5 * (12.7 - 4) / 2
Out[2]: 21.75
```

Python uses the asterisk (`*`) for multiplication and the forward slash (`/`) for division. As in mathematics, parentheses force the evaluation order, so the parenthesized expression `(12.7 - 4)` evaluates first, giving `8.7`. Next, `5 * 8.7` evaluates giving `43.5`. Then, `43.5 / 2` evaluates, giving the result `21.75`, which IPython displays in `Out[2]`. Whole numbers, like `5`, `4` and `2`, are called **integers**. Numbers with decimal points, like `12.7`, `43.5` and `21.75`, are called **floating-point numbers**.

Exiting Interactive Mode

To leave interactive mode, you can:

- Type the `exit` command at the current `In []` prompt and press *Enter* to exit immediately.
- Type the key sequence *<Ctrl>* + *d* (or *<control>* + *d*). This displays the prompt `"Do you really want to exit ([y]/n)?"`. The square brackets around `y` indicate that it's the default response—pressing *Enter* submits the default response and exits.
- Type *<Ctrl>* + *d* (or *<control>* + *d*) twice (macOS and Linux only).

✓ Self Check

1 *(Fill-In)* In IPython interactive mode, you'll enter small bits of Python code called ________ and immediately see their results.
Answer: snippets.

2 In IPython ________ mode, you'll execute Python code loaded from a file that has the `.py` extension (short for Python).
Answer: script.

3 *(IPython Session)* Evaluate the expression `5 * (3 + 4)` both with and without the parentheses. Do you get the same result? Why or why not?

27. In the next chapter, you'll see that there are some cases in which `Out[]` is not displayed.

Answer: You get different results because snippet [1] first calculates 3 + 4, which is 7, then multiplies that by 5. Snippet [2] first multiplies 5 * 3, which is 15, then adds that to 4.

```
In [1]: 5 * (3 + 4)
Out[1]: 35

In [2]: 5 * 3 + 4
Out[2]: 19
```

1.10.2 Executing a Python Program Using the IPython Interpreter

In this section, you'll execute a script named `RollDieDynamic.py` that you'll write in Chapter 6. The **`.py` extension** indicates that the file contains Python source code. The script `RollDieDynamic.py` simulates rolling a six-sided die. It presents a colorful animated visualization that dynamically graphs the frequencies of each die face.

Changing to This Chapter's Examples Folder

You'll find the script in the book's `ch01` source-code folder. In the Before You Begin section you extracted the `examples` folder to your user account's `Documents` folder. Each chapter has a folder containing that chapter's source code. The folder is named `ch##`, where `##` is a two-digit chapter number from `01` to `17`. First, open your system's command-line window. Next, use the `cd` ("change directory") command to change to the `ch01` folder:

- On macOS/Linux, type `cd ~/Documents/examples/ch01`, then press *Enter*.
- On Windows, type `cd C:\Users\`*YourAccount*`\Documents\examples\ch01`, then press *Enter*.

Executing the Script

To execute the script, type the following command at the command line, then press *Enter*:

```
ipython RollDieDynamic.py 6000 1
```

The script displays a window, showing the visualization. The numbers `6000` and `1` tell this script the number of times to roll dice and how many dice to roll each time. In this case, we'll update the chart `6000` times for `1` die at a time.

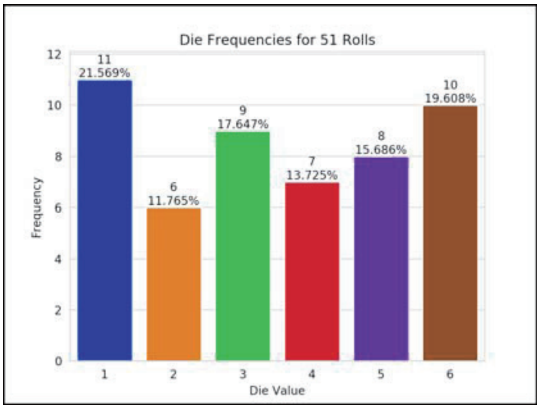
For a six-sided die, the values 1 through 6 should each occur with "equal likelihood"—the probability of each is 1/6th or about 16.667%. If we roll a die 6000 times, we'd expect about 1000 of each face. Like coin tossing, die rolling is *random*, so there could be some faces with fewer than 1000, some with 1000 and some with more than 1000. We took the screen captures on the next page during the script's execution. This script uses randomly generated die values, so your results will differ. Experiment with the script by changing the value 1 to 100, 1000 and 10000. Notice that as the number of die rolls gets larger, the frequencies zero in on 16.667%. This is a phenomenon of the "Law of Large Numbers."

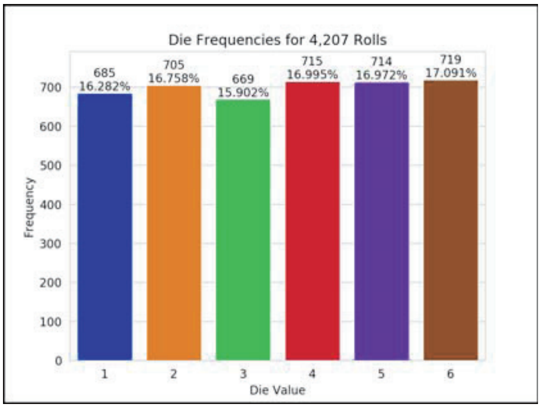
Creating Scripts

Typically, you create your Python source code in an editor that enables you to type text. Using the editor, you type a program, make any necessary corrections and save it to your computer. **Integrated development environments (IDEs)** provide tools that support the entire software-development process, such as editors, debuggers for locating **logic errors** that cause programs to execute incorrectly and more. Some popular Python IDEs include Spyder (which comes with Anaconda), PyCharm and Visual Studio Code.

Roll the dice 6000 times and roll 1 die each time:

```
ipython RollDieDynamic.py 6000 1
```





Problems That May Occur at Execution Time

Programs often do not work on the first try. For example, an executing program might try to divide by zero (an illegal operation in Python). This would cause the program to display an error message. If this occurred in a script, you'd return to the editor, make the necessary corrections and re-execute the script to determine whether the corrections fixed the problem(s).

Errors such as division by zero occur as a program runs, so they're called **runtime errors** or **execution-time errors**. **Fatal runtime errors** cause programs to terminate immediately without having successfully performed their jobs. **Non-fatal runtime errors** allow programs to run to completion, often producing incorrect results.

✓ Self Check

1 *(Discussion)* When the example in this section finishes all 6000 rolls, does the chart show that the die faces appeared *about* 1000 times each?

Answer: Most likely, yes. This example is based on random-number generation, so the results may vary. Because of this randomness, most of the counts will be a little more than 1000 or a little less.

2 *(Discussion)* Run the example in this section again. Do the faces appear the same number of times as they did in the previous execution?

Answer: Probably not. This example uses random-number generation, so successive executions likely will produce different results. In Chapter 4, we'll show how to force Python to produce the same sequence of random numbers. This is important for *reproducibility*—a crucial data-science topic you'll investigate in the chapter exercises and throughout the book. You'll want other data scientists to be able to reproduce your results. Also, you'll want to be able to reproduce your own experimental results. This is helpful when you find and fix an error in your program and want to make sure that you've corrected it properly.

1.10.3 Writing and Executing Code in a Jupyter Notebook

The Anaconda Python Distribution that you installed in the Before You Begin section comes with the **Jupyter Notebook**—an interactive, browser-based environment in which you can write and execute code and intermix the code with text, images and video. Jupyter Notebooks are broadly used in the data-science community in particular and the broader scientific community in general. They're the preferred means of doing Python-based data analytics studies and *reproducibly* communicating their results. The Jupyter Notebook environment actually supports many programming languages.

For your convenience, all of the book's source code also is provided in Jupyter Notebooks that you can simply load and execute. In this section, you'll use the **JupyterLab** interface, which enables you to manage your notebook files and other files that your notebooks use (like images and videos). As you'll see, JupyterLab also makes it convenient to write code, execute it, see the results, modify the code and execute it again.

You'll see that coding in a Jupyter Notebook is similar to working with IPython—in fact, Jupyter Notebooks use IPython by default. In this section, you'll create a notebook, add the code from Section 1.10.1 to it and execute that code.

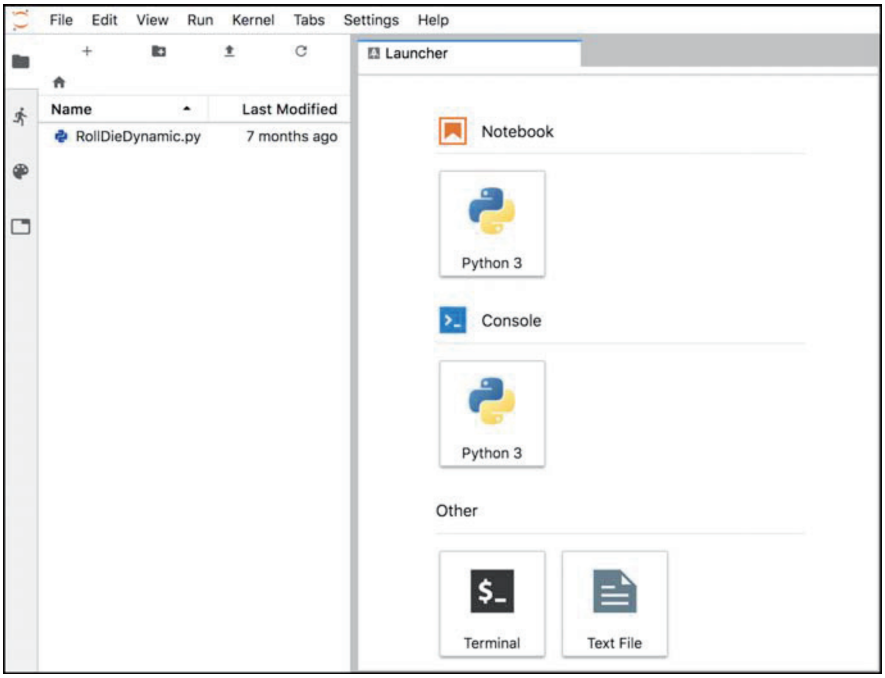
Opening JupyterLab in Your Browser

To open JupyterLab, change to the `ch01` examples folder in your Terminal, shell or Anaconda Command Prompt (as in Section 1.10.2), type the following command, then press *Enter* (or *Return*):

```
jupyter lab
```

This executes the Jupyter Notebook server on your computer and opens JupyterLab in your default web browser, showing the `ch01` folder's contents in the **File Browser** tab

at the left side of the JupyterLab interface:



The Jupyter Notebooks server enables you to load and run Jupyter Notebooks in your web browser. From the JupyterLab **Files** tab, you can double-click files to open them in the right side of the window where the **Launcher** tab is currently displayed. Each file you open appears as a separate tab in this part of the window. If you accidentally close your browser, you can reopen JupyterLab by entering the following address in your web browser

```
http://localhost:8888/lab
```

Creating a New Jupyter Notebook

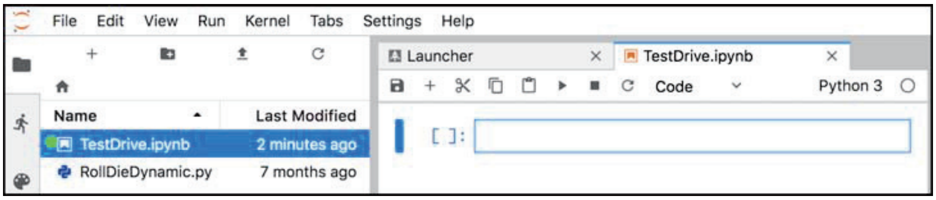
In the **Launcher** tab under **Notebook**, click the **Python 3** button to create a new Jupyter Notebook named `Untitled.ipynb` in which you can enter and execute Python 3 code. The file extension **`.ipynb`** is short for IPython Notebook—the original name of the Jupyter Notebook.

Renaming the Notebook

Rename `Untitled.ipynb` as `TestDrive.ipynb`:

1. Right-click the `Untitled.ipynb` tab and select **Rename Notebook...**.
2. Change the name to `TestDrive.ipynb` and click **RENAME**.

The top of JupyterLab should now appear as follows:



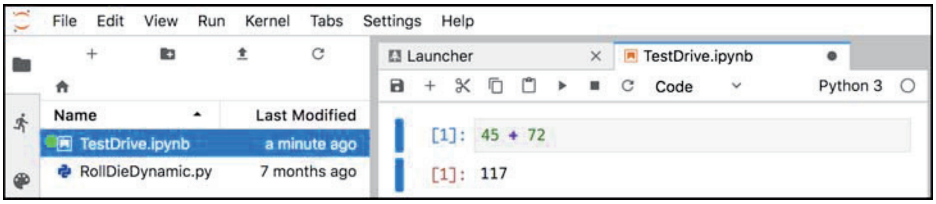
Name	Last Modified
TestDrive.ipynb	2 minutes ago
RollDieDynamic.py	7 months ago

Evaluating an Expression

The unit of work in a notebook is a **cell** in which you can enter code snippets. By default, a new notebook contains one cell—the rectangle in the `TestDrive.ipynb` notebook—but you can add more. To the cell's left, the notation `[ ]:` is where the Jupyter Notebook will display the cell's snippet number *after* you execute the cell. Click in the cell, then type the expression

```
45 + 72
```

To execute the current cell's code, type *Ctrl* + *Enter* (or *control* + *Enter*). JupyterLab executes the code in IPython, then displays the results below the cell:



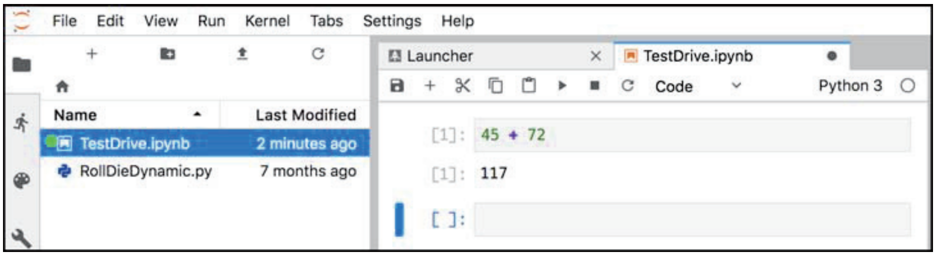
Name	Last Modified
TestDrive.ipynb	a minute ago
RollDieDynamic.py	7 months ago

```
[1]: 45 + 72

[1]: 117
```

Adding and Executing Another Cell

Let's evaluate a more complex expression. First, click the **+** button in the toolbar above the notebook's first cell—this adds a new cell below the current one:

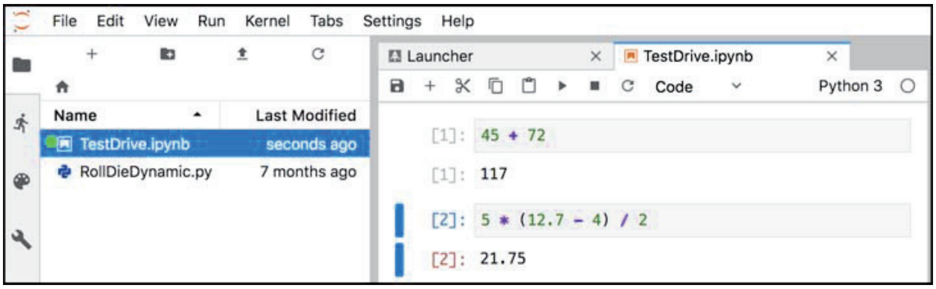


```
[1]: 45 + 72
[1]: 117
[ ]:
```

Click in the new cell, then type the expression

```
5 * (12.7 - 4) / 2
```

and execute the cell by typing *Ctrl* + *Enter* (or *control* + *Enter*):



```
[1]: 45 + 72
[1]: 117
[2]: 5 * (12.7 - 4) / 2
[2]: 21.75
```

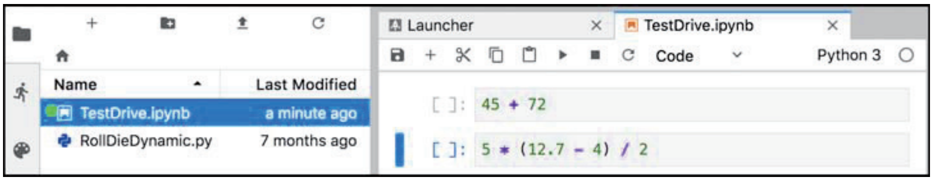
Saving the Notebook

If your notebook has unsaved changes, the **X** in the notebook's tab will change to ●. To save the notebook, select the **File** menu in JupyterLab (not at the top of your browser's window), then select **Save Notebook**.

Notebooks Provided with Each Chapter's Examples

For your convenience, each chapter's examples also are provided as ready-to-execute notebooks without their outputs. This enables you to work through them snippet-by-snippet and see the outputs appear as you execute each snippet.

So that we can show you how to load an existing notebook and execute its cells, let's reset the `TestDrive.ipynb` notebook to remove its output and snippet numbers. This will return it to a state like the notebooks we provide for the subsequent chapters' examples. From the **Kernel** menu select **Restart Kernel and Clear All Outputs...**, then click the **RESTART** button. The preceding command also is helpful whenever you wish to re-execute a notebook's snippets. The notebook should now appear as follows:



```
[ ]: 45 + 72
[ ]: 5 * (12.7 - 4) / 2
```

From the **File** menu, select **Save Notebook**, then click the `TestDrive.ipynb` tab's **X** button to close the notebook.

Opening and Executing an Existing Notebook

When you launch JupyterLab from a given chapter's examples folder, you'll be able to open notebooks from that folder or any of its subfolders. Once you locate a specific notebook, double-click it to open it. Open the `TestDrive.ipynb` notebook again now. Once a notebook is open, you can execute each cell individually, as you did earlier in this section, or you can execute the entire notebook at once. To do so, from the **Run** menu select **Run All Cells**. The notebook will execute the cells in order, displaying each cell's output below that cell.

Closing JupyterLab

When you're done with JupyterLab, you can close its browser tab, then in the Terminal, shell or Anaconda Command Prompt from which you launched JupyterLab, type *Ctrl* + *c* (or *control* + *c*) twice.

JupyterLab Tips

While working in JupyterLab, you might find these tips helpful:

- If you need to enter and execute many snippets, you can execute the current cell *and* add a new one below it by typing *Shift* + *Enter*, rather than *Ctrl* + *Enter* (or *control* + *Enter*).
- As you get into the later chapters, some of the snippets you'll enter in Jupyter Notebooks will contain many lines of code. To display line numbers within each cell, select **Show line numbers** from JupyterLab's **View** menu.

More Information on Working with JupyterLab

JupyterLab has many more features that you'll find helpful. We recommend that you read the Jupyter team's introduction to JupyterLab at:

```
https://jupyterlab.readthedocs.io/en/stable/index.html
```

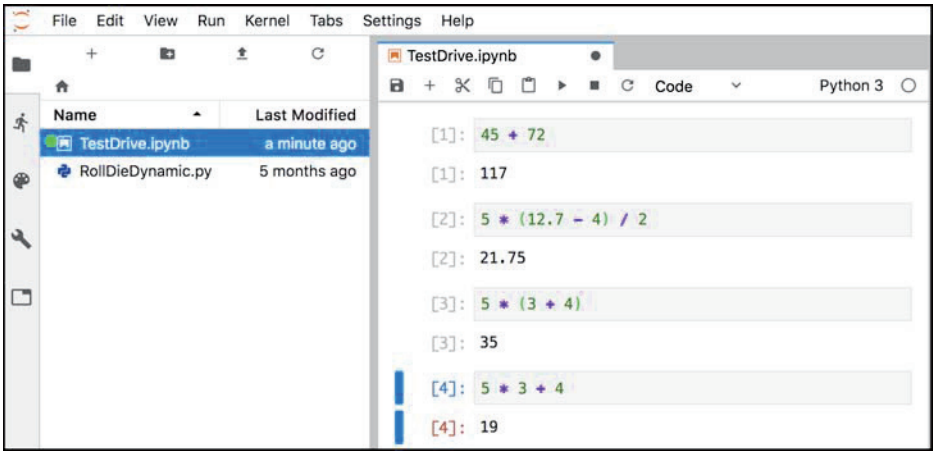
For a quick overview, click **Overview** under **GETTING STARTED**. Also, under **USER GUIDE** read the introductions to **The JupyterLab Interface**, **Working with Files**, **Text Editor** and **Notebooks** for many additional features.

✓ Self Check

1 *(True/False)* Jupyter Notebooks are the preferred means of doing Python-based data analytics studies and reproducibly communicating their results.
Answer: True.

2 *(Jupyter Notebook Session)* Ensure that JupyterLab is running, then open your `TestDrive.ipynb` notebook. Add and execute two more snippets that evaluate the expression `5 * (3 + 4)` both with and without the parentheses. You should see the same results as in Section 1.10.1's Self Check Exercise 3.

Answer:


File Edit View Run Kernel Tabs Settings Help
Name
Last Modified
TestDrive.ipynb a minute ago
RollDieDynamic.py 5 months ago
TestDrive.ipynb
Code
Python 3

```
[1]: 45 + 72
[1]: 117
[2]: 5 * (12.7 - 4) / 2
[2]: 21.75
[3]: 5 * (3 + 4)
[3]: 35
[4]: 5 * 3 + 4
[4]: 19
```

1.11 Internet and World Wide Web

In the late 1960s, ARPA—the Advanced Research Projects Agency of the United States Department of Defense—rolled out plans for networking the main computer systems of approximately a dozen ARPA-funded universities and research institutions. The computers were to be connected with communications lines operating at speeds on the order of 50,000 bits per second, a stunning rate at a time when most people (of the few who even had networking access) were connecting over telephone lines to computers at a rate of 110 bits per second. Academic research was about to take a giant leap forward. ARPA proceeded to implement what quickly became known as the ARPANET, the precursor to today's **Internet**. Today's fastest Internet speeds are on the order of billions of bits per second with trillion-bits-per-second (terabit) speeds already being tested![28]

Things worked out differently from the original plan. Although the ARPANET enabled researchers to network their computers, its main benefit proved to be the capability for quick and easy communication via what came to be known as electronic mail (e-mail). This is true even on today's Internet, with e-mail, instant messaging, file transfer and social media such as Snapchat, Instagram, Facebook and Twitter enabling billions of people worldwide to communicate quickly and easily.

The protocol (set of rules) for communicating over the ARPANET became known as the **Transmission Control Protocol (TCP)**. TCP ensured that messages, consisting of sequentially numbered pieces called *packets*, were properly delivered from sender to receiver, arrived intact and were assembled in the correct order.

1.11.1 Internet: A Network of Networks

In parallel with the early evolution of the Internet, organizations worldwide were implementing their own networks for both intra-organization (that is, within an organization) and inter-organization (that is, between organizations) communication. A huge variety of networking hardware and software appeared. One challenge was to enable these different

28. `https://testinternetspeed.org/blog/bt-testing-1-4-terabit-internet-connections/`.

networks to communicate with each other. ARPA accomplished this by developing the **Internet Protocol (IP)**, which created a true "network of networks," the current architecture of the Internet. The combined set of protocols is now called **TCP/IP**. Each Internet-connected device has an **IP address**—a unique numerical identifier used by devices communicating via TCP/IP to locate one another on the Internet.

Businesses rapidly realized that by using the Internet, they could improve their operations and offer new and better services to their clients. Companies started spending large amounts of money to develop and enhance their Internet presence. This generated fierce competition among communications carriers and hardware and software suppliers to meet the increased infrastructure demand. As a result, **bandwidth**—the information-carrying capacity of communications lines—on the Internet has increased tremendously, while hardware costs have plummeted.

1.11.2 World Wide Web: Making the Internet User-Friendly

The **World Wide Web** (simply called "the web") is a collection of hardware and software associated with the Internet that allows computer users to locate and view documents (with various combinations of text, graphics, animations, audios and videos) on almost any subject. In 1989, Tim Berners-Lee of CERN (the European Organization for Nuclear Research) began developing **HyperText Markup Language (HTML)**—the technology for sharing information via "hyperlinked" text documents. He also wrote communication protocols such as **HyperText Transfer Protocol (HTTP)** to form the backbone of his new hypertext information system, which he referred to as the World Wide Web.

In 1994, Berners-Lee founded the **World Wide Web Consortium** (**W3C**, `https://www.w3.org`), devoted to developing web technologies. One of the W3C's primary goals is to make the web universally accessible to everyone regardless of disabilities, language or culture.

1.11.3 The Cloud

More and more computing today is done "in the cloud"—that is, distributed across the Internet worldwide. The apps you use daily are heavily dependent on various **cloud-based services** that use massive clusters of computing resources (computers, processors, memory, disk drives, etc.) and databases that communicate over the Internet with each other and the apps you use. A service that provides access to itself over the Internet is known as a **web service**. As you'll see, using cloud-based services in Python often is as simple as creating a software object and interacting with it. That object then uses web services that connect to the cloud on your behalf.

Throughout the Chapters 12–17 examples and exercises, you'll work with many cloud-based services:

- In Chapters 13 and 17, you'll use Twitter's web services (via the Python library Tweepy) to get information about specific Twitter users, search for tweets from the last seven days and to receive streams of tweets as they occur—that is, in **real time**.
- In Chapters 12 and 13, you'll use the Python library TextBlob to translate text between languages. Behind the scenes, TextBlob uses the Google Translate web service to perform those translations.

- In Chapter 14, you'll use the IBM Watson's Text to Speech, Speech to Text and Translate services. You'll implement a traveler's assistant translation app that enables you to speak a question in English, transcribes the speech to text, translates the text to Spanish and speaks the Spanish text. The app then allows you to speak a Spanish response (in case you don't speak Spanish, we provide an audio file you can use), transcribes the speech to text, translates the text to English and speaks the English response. Via IBM Watson demos, you'll also experiment with many other Watson cloud-based services in Chapter 14.
- In Chapter 17, you'll work with Microsoft Azure's HDInsight service and other Azure web services as you learn to implement big-data applications using Apache Hadoop and Spark. Azure is Microsoft's set of cloud-based services.
- In Chapter 17, you'll use the Dweet.io web service to simulate an Internet-connected thermostat that publishes temperature readings online. You'll also use a web-based service to create a "dashboard" that visualizes the temperature readings over time and warns you if the temperature gets too low or too high.
- In Chapter 17, you'll use a web-based dashboard to visualize a simulated stream of live sensor data from the PubNub web service. You'll also create a Python app that visualizes a PubNub simulated stream of live stock-price changes.
- In multiple exercises, you'll research, explore and use Wikipedia web services.

In most cases, you'll create Python objects that interact with web services on your behalf, hiding the details of how to access these services over the Internet.

Mashups

The applications-development methodology of *mashups* enables you to rapidly develop powerful software applications by combining (often free) complementary web services and other forms of information feeds—as you'll do in our IBM Watson traveler's assistant translation app. One of the first mashups combined the real-estate listings provided by `http://www.craigslist.org` with the mapping capabilities of Google Maps to offer maps that showed the locations of homes for sale or rent in a given area.

ProgrammableWeb (`http://www.programmableweb.com/`) provides a directory of over 20,750 web services and almost 8,000 mashups. They also provide how-to guides and sample code for working with web services and creating your own mashups. According to their website, some of the most widely used web services are Facebook, Google Maps, Twitter and YouTube.

1.11.4 Internet of Things

The Internet is no longer just a network of *computers*—it's an **Internet of Things (IoT)**. A *thing* is any object with an IP address and the ability to send, and in some cases receive, data automatically over the Internet. Such *things* include:

- a car with a transponder for paying tolls,
- monitors for parking-space availability in a garage,
- a heart monitor implanted in a human,
- water quality monitors,

- a smart meter that reports energy usage,
- radiation detectors,
- item trackers in a warehouse,
- mobile apps that can track your movement and location,
- smart thermostats that adjust room temperatures based on weather forecasts and activity in the home, and
- intelligent home appliances.

According to `statista.com`, there are already over 23 billion IoT devices in use today, and there could be over 75 billion IoT devices in 2025.[29]

✓ Self Check for Section 1.11

1 *(Fill-In)* The ________ was the precursor to today's Internet.
Answer: ARPANET.

2 *(Fill-In)* The ________ (simply called "the web") is a collection of hardware and software associated with the Internet that allows computer users to locate and view documents (with various combinations of text, graphics, animations, audios and videos).
Answer: World Wide Web.

3 *(Fill-In)* In the Internet of Things (IoT), a *thing* is any object with a(n) ________ and the ability to send, and in some cases receive, data automatically over the Internet.
Answer: IP address.

1.12 Software Technologies

As you learn about and work in software development, you'll frequently encounter the following buzzwords:

- **Refactoring**: Reworking programs to make them clearer and easier to maintain while preserving their correctness and functionality. Many IDEs contain built-in *refactoring tools* to do major portions of the reworking automatically.
- **Design patterns**: Proven architectures for constructing flexible and maintainable object-oriented software. The field of design patterns tries to enumerate those recurring patterns, encouraging software designers to *reuse* them to develop better-quality software using less time, money and effort.
- **Cloud computing**: You can use software and data stored in the "cloud"—i.e., accessed on remote computers (or servers) via the Internet and available on demand—rather than having it stored locally on your desktop, notebook computer or mobile device. This allows you to increase or decrease computing resources to meet your needs at any given time, which is more cost effective than purchasing hardware to provide enough storage and processing power to meet occasional peak demands. Cloud computing also saves money by shifting to the

29. `https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/`.

service provider the burden of managing these apps (such as installing and upgrading the software, security, backups and disaster recovery).

- **Software Development Kits (SDKs)**—The tools and documentation that developers use to program applications. For example, in Chapter 14, you'll use the Watson Developer Cloud Python SDK to interact with IBM Watson services from a Python application.

✓ Self Check

1 *(Fill-In)* ________ is the process of reworking programs to make them clearer and easier to maintain while preserving their correctness and functionality.
Answer: refactoring.

1.13 How Big Is Big Data?

For computer scientists and data scientists, data is now as important as writing programs. According to IBM, approximately 2.5 quintillion bytes (2.5 *exabytes*) of data are created daily,[30] and 90% of the world's data was created in the last two years.[31] According to IDC, the global data supply will reach 175 *zettabytes* (equal to 175 trillion gigabytes or 175 billion terabytes) annually by 2025.[32] Consider the following examples of various popular data measures.

Megabytes (MB)

One megabyte is about one million (actually 2^{20}) bytes. Many of the files we use on a daily basis require one or more MBs of storage. Some examples include:

- MP3 audio files—High-quality MP3s range from 1 to 2.4 MB per minute.[33]
- Photos—JPEG format photos taken on a digital camera can require about 8 to 10 MB per photo.
- Video—Smartphone cameras can record video at various resolutions. Each minute of video can require many megabytes of storage. For example, on one of our iPhones, the **Camera** settings app reports that 1080p video at 30 frames-per-second (FPS) requires 130 MB/minute and 4K video at 30 FPS requires 350 MB/minute.

Gigabytes (GB)

One gigabyte is about 1000 megabytes (actually 2^{30} bytes). A dual-layer DVD can store up to 8.5 GB[34], which translates to:

- as much as 141 hours of MP3 audio,
- approximately 1000 photos from a 16-megapixel camera,

30. https://www.ibm.com/blogs/watson/2016/06/welcome-to-the-world-of-a-i/.
31. https://public.dhe.ibm.com/common/ssi/ecm/wr/en/wrl12345usen/watson-customer-engagement-watson-marketing-wr-other-papers-and-reports-wrl12345usen-20170719.pdf.
32. https://www.networkworld.com/article/3325397/storage/idc-expect-175-zettabytes-of-data-worldwide-by-2025.html.
33. https://www.audiomountain.com/tech/audio-file-size.html.
34. https://en.wikipedia.org/wiki/DVD.

- approximately 7.7 minutes of 1080p video at 30 FPS, or
- approximately 2.85 minutes of 4K video at 30 FPS.

The current highest-capacity Ultra HD Blu-ray discs can store up to 100 GB of video.[35] Streaming a 4K movie can use between 7 and 10 GB per hour (highly compressed).

Terabytes (TB)

One terabyte is about 1000 gigabytes (actually 2^{40} bytes). Recent disk drives for desktop computers come in sizes up to 15 TB,[36] which is equivalent to:

- approximately 28 years of MP3 audio,
- approximately 1.68 million photos from a 16-megapixel camera,
- approximately 226 hours of 1080p video at 30 FPS and
- approximately 84 hours of 4K video at 30 FPS.

Nimbus Data now has the largest solid-state drive (SSD) at 100 TB, which can store 6.67 times the 15-TB examples of audio, photos and video listed above.[37]

Petabytes, Exabytes and Zettabytes

There are nearly four billion people online creating about 2.5 quintillion bytes of data each day[38]—that's 2500 petabytes (each petabyte is about 1000 terabytes) or 2.5 exabytes (each exabyte is about 1000 petabytes). According to a March 2016 *AnalyticsWeek* article, within five years there will be over 50 billion devices connected to the Internet (most of them through the Internet of Things, which we discuss in Sections 1.11.4 and 17.8) and by 2020 we'll be producing 1.7 megabytes of new data every second *for every person on the planet*.[39] At today's numbers (approximately 7.7 billion people[40]), that's about

- 13 petabytes of new data per second,
- 780 petabytes per minute,
- 46,800 petabytes (46.8 exabytes) per hour and
- 1,123 exabytes per day—that's 1.123 zettabytes (ZB) per day (each zettabyte is about 1000 exabytes).

That's the equivalent of over 5.5 million hours (over 600 years) of 4K video every day or approximately 116 billion photos every day!

Additional Big-Data Stats

For a real-time sense of big data, check out `https://www.internetlivestats.com`, with various statistics, including the numbers so far today of

- Google searches.

35. `https://en.wikipedia.org/wiki/Ultra_HD_Blu-ray`.
36. `https://www.zdnet.com/article/worlds-biggest-hard-drive-meet-western-digitals-15tb-monster/`.
37. `https://www.cinema5d.com/nimbus-data-100tb-ssd-worlds-largest-ssd/`.
38. `https://public.dhe.ibm.com/common/ssi/ecm/wr/en/wrl12345usen/watson-customer-engagement-watson-marketing-wr-other-papers-and-reports-wrl12345usen-20170719.pdf`.
39. `https://analyticsweek.com/content/big-data-facts/`.
40. `https://en.wikipedia.org/wiki/World_population`.

- Tweets.
- Videos viewed on YouTube.
- Photos uploaded on Instagram.

You can click each statistic to drill down for more information. For instance, they say over 250 *billion* tweets have been sent in 2018.

Some other interesting big-data facts:

- Every hour, YouTube users upload 24,000 hours of video, and almost 1 billion hours of video are watched on YouTube every day.[41]
- Every second, there are 51,773 GBs (or 51.773 TBs) of Internet traffic, 7894 tweets sent, 64,332 Google searches and 72,029 YouTube videos viewed.[42]
- On Facebook each day there are 800 million "**likes**,"[43] 60 million emojis are sent,[44] and there are over two billion searches of the more than 2.5 trillion Facebook posts since the site's inception.[45]
- In June 2017, Will Marshall, CEO of Planet, said the company has 142 satellites that image the whole planet's land mass once per day. They add one million images and seven TBs of new data each day. Together with their partners, they're using machine learning on that data to improve crop yields, see how many ships are in a given port and track deforestation. With respect to Amazon deforestation, he said: "Used to be we'd wake up after a few years and there's a big hole in the Amazon. Now we can literally count every tree on the planet every day."[46]

Domo, Inc. has a nice infographic called "Data Never Sleeps 6.0" showing how much data is generated *every minute*, including:[47]

- 473,400 tweets sent.
- 2,083,333 Snapchat photos shared.
- 97,222 hours of Netflix video viewed.
- 12,986,111 million text messages sent.
- 49,380 Instagram posts.
- 176,220 Skype calls.
- 750,000 Spotify songs streamed.
- 3,877,140 Google searches.
- 4,333,560 YouTube videos watched.

41. `https://www.brandwatch.com/blog/youtube-stats/`.
42. `http://www.internetlivestats.com/one-second`.
43. `https://newsroom.fb.com/news/2017/06/two-billion-people-coming-together-on-facebook`.
44. `https://mashable.com/2017/07/17/facebook-world-emoji-day/`.
45. `https://techcrunch.com/2016/07/27/facebook-will-make-you-talk/`.
46. `https://www.bloomberg.com/news/videos/2017-06-30/learning-from-planet-s-shoe-boxed-sized-satellites-video`, June 30, 2017.
47. `https://www.domo.com/learn/data-never-sleeps-6`.

Computing Power Over the Years

Data is getting more massive and so is the computing power for processing it. The performance of today's processors is often measured in terms of **FLOPS (floating-point operations per second)**. In the early to mid-1990s, the fastest supercomputer speeds were measured in gigaflops (10^9 FLOPS). By the late 1990s, Intel produced the first teraflop (10^{12} FLOPS) supercomputers. In the early-to-mid 2000s, speeds reached hundreds of teraflops, then in 2008, IBM released the first petaflop (10^{15} FLOPS) supercomputer. Currently, the fastest supercomputer—the IBM Summit, located at the Department of Energy's (DOE) Oak Ridge National Laboratory (ORNL)—is capable of 122.3 petaflops.[48]

Distributed computing can link thousands of personal computers via the Internet to produce even more FLOPS. In late 2016, the Folding@home network—a distributed network in which people volunteer their personal computers' resources for use in disease research and drug design[49]—was capable of over 100 petaflops.[50] Companies like IBM are now working toward supercomputers capable of exaflops (10^{18} FLOPS).[51]

The **quantum computers** now under development theoretically could operate at 18,000,000,000,000,000,000 times the speed of today's "conventional computers"![52] This number is so extraordinary that in one second, a quantum computer theoretically could do staggeringly more calculations than the total that have been done by all computers since the world's first computer appeared. This almost unimaginable computing power could wreak havoc with blockchain-based cryptocurrencies like Bitcoin. Engineers are already rethinking blockchain to prepare for such massive increases in computing power.[53]

The history of supercomputing power is that it eventually works its way down from research labs, where extraordinary amounts of money have been spent to achieve those performance numbers, into "reasonably priced" commercial computer systems and even desktop computers, laptops, tablets and smartphones.

Computing power's cost continues to decline, especially with cloud computing. People used to ask the question, "How much computing power do I need on my system to deal with my *peak* processing needs?" Today, that thinking has shifted to "Can I quickly carve out on the cloud what I need *temporarily* for my most demanding computing chores?" You pay for only what you use to accomplish a given task.

Processing the World's Data Requires Lots of Electricity

Data from the world's Internet-connected devices is exploding, and processing that data requires tremendous amounts of energy. According to a recent article, energy use for processing data in 2015 was growing at 20% per year and consuming approximately three to five percent of the world's power. The article says that total data-processing power consumption could reach 20% by 2025.[54]

48. `https://en.wikipedia.org/wiki/FLOPS`.
49. `https://en.wikipedia.org/wiki/Folding@home`.
50. `https://en.wikipedia.org/wiki/FLOPS`.
51. `https://www.ibm.com/blogs/research/2017/06/supercomputing-weather-model-exascale/`.
52. `https://medium.com/@n.biedrzycki/only-god-can-count-that-fast-the-world-of-quantum-computing-406a0a91fcf4`.
53. `https://singularityhub.com/2017/11/05/is-quantum-computing-an-existential-threat-to-blockchain-technology/`.
54. `https://www.theguardian.com/environment/2017/dec/11/tsunami-of-data-could-consume-fifth-global-electricity-by-2025`.

Another enormous electricity consumer is the blockchain-based cryptocurrency Bitcoin. Processing just one Bitcoin transaction uses approximately the same amount of energy as powering the average American home for a week. The energy use comes from the process Bitcoin "miners" use to prove that transaction data is valid.[55]

According to some estimates, a year of Bitcoin transactions consumes more energy than many countries.[56] Together, Bitcoin and Ethereum (another popular blockchain-based platform and cryptocurrency) consume more energy per year than Israel and almost as much as Greece.[57]

Morgan Stanley predicted in 2018 that "the electricity consumption required to create cryptocurrencies this year could actually outpace the firm's projected global electric vehicle demand—in 2025."[58] This situation is unsustainable, especially given the huge interest in blockchain-based applications, even beyond the cryptocurrency explosion. The blockchain community is working on fixes.[59,60]

Big-Data Opportunities

The big-data explosion is likely to continue exponentially for years to come. With 50 billion computing devices on the horizon, we can only imagine how many more there will be over the next few decades. It's crucial for businesses, governments, the military, and even individuals to get a handle on all this data.

It's interesting that some of the best writings about big data, data science, artificial intelligence and more are coming out of distinguished business organizations, such as J.P. Morgan, McKinsey and more. Big data's appeal to big business is undeniable given the rapidly accelerating accomplishments. Many companies are making significant investments and getting valuable results through technologies in this book, such as big data, machine learning, deep learning, and natural-language processing. This is forcing competitors to invest as well, rapidly increasing the need for computing professionals with data-science and computer science experience. This growth is likely to continue for many years.

✓ Self Check

1 *(Fill-In)* Today's processor performance is often measured in terms of ________.
Answer: FLOPS (floating-point operations per second).

2 *(Fill-In)* The technology that could wreak havoc with blockchain-based cryptocurrencies, like Bitcoin, and other blockchain-based technologies is ________.
Answer: quantum computers.

3 *(True/False)* With cloud computing you pay a fixed price for cloud services regardless of how much you use those services?
Answer: False. A key cloud-computing benefit is that you pay for only what you use to accomplish a given task.

55. `https://motherboard.vice.com/en_us/article/ywbbpm/bitcoin-mining-electricity-consumption-ethereum-energy-climate-change`.
56. `https://digiconomist.net/bitcoin-energy-consumption`.
57. `https://digiconomist.net/ethereum-energy-consumption`.
58. `https://www.morganstanley.com/ideas/cryptocurrencies-global-utilities`.
59. `https://www.technologyreview.com/s/609480/bitcoin-uses-massive-amounts-of-energy-but-theres-a-plan-to-fix-it/`.
60. `http://mashable.com/2017/12/01/bitcoin-energy/`.

1.13.1 Big Data Analytics

Data analytics is a mature and well-developed academic and professional discipline. The term “data analysis” was coined in 1962,[61] though people have been analyzing data using statistics for thousands of years going back to the ancient Egyptians.[62] Big data analytics is a more recent phenomenon—the term “big data” was coined around 2000.[63]

Consider four of the V’s of big data[64,65]:

1. Volume—the amount of data the world is producing is growing exponentially.
2. Velocity—the speed at which that data is being produced, the speed at which it moves through organizations and the speed at which data changes are growing quickly.[66,67,68]
3. Variety—data used to be alphanumeric (that is, consisting of alphabetic characters, digits, punctuation and some special characters)—today it also includes images, audios, videos and data from an exploding number of Internet of Things sensors in our homes, businesses, vehicles, cities and more.
4. Veracity—the validity of the data—is it complete and accurate? Can we trust that data when making crucial decisions? Is it real?

Most data is now being created digitally in a *variety* of types, in extraordinary *volumes* and moving at astonishing *velocities*. Moore’s Law and related observations have enabled us to store data economically and to process and move it faster—and all at rates growing exponentially over time. Digital data storage has become so vast in capacity, cheap and small that we can now conveniently and economically retain *all* the digital data we’re creating.[69] That’s big data.

The following Richard W. Hamming quote—although from 1962—sets the tone for the rest of this book:

“The purpose of computing is insight, not numbers.”[70]

Data science is producing new, deeper, subtler and more valuable insights at a remarkable pace. It’s truly making a difference. Big data analytics is an integral part of the answer. We address big data infrastructure in Chapter 17 with hands-on case studies on NoSQL data-

61. `https://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/`.
62. `https://www.flydata.com/blog/a-brief-history-of-data-analysis/`.
63. `https://bits.blogs.nytimes.com/2013/02/01/the-origins-of-big-data-an-etymological-detective-story/`.
64. `https://www.ibmbigdatahub.com/infographic/four-vs-big-data`.
65. There are lots of articles and papers that add many other “V-words” to this list.
66. `https://www.zdnet.com/article/volume-velocity-and-variety-understanding-the-three-vs-of-big-data/`.
67. `https://whatis.techtarget.com/definition/3Vs`.
68. `https://www.forbes.com/sites/brentdykes/2017/06/28/big-data-forget-volume-and-variety-focus-on-velocity`.
69. `http://www.lesk.com/mlesk/ksg97/ksg.html`. [The following article pointed us to this Michael Lesk article: `https://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/`.]
70. Hamming, R. W., *Numerical Methods for Scientists and Engineers* (New York, NY., McGraw Hill, 1962). [The following article pointed us to Hamming’s book and his quote that we cited: `https://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/`.]

bases, Hadoop MapReduce programming, Spark, real-time Internet of Things (IoT) stream programming and more.

To get a sense of big data's scope in industry, government and academia, check out the high-resolution graphic.[71] You can click to zoom for easier readability:

```
http://mattturck.com/wp-content/uploads/2018/07/
   Matt_Turck_FirstMark_Big_Data_Landscape_2018_Final.png
```

1.13.2 Data Science and Big Data Are Making a Difference: Use Cases

The data-science field is growing rapidly because it's producing significant results that are making a difference. We enumerate data-science and big data use cases in the following table. We expect that the use cases and our examples, exercises and projects will inspire interesting term projects, directed-study projects, capstone-course projects and thesis research. Big-data analytics has resulted in improved profits, better customer relations, and even sports teams winning more games and championships while spending less on players.[72,73,74]

Data-science use cases		
anomaly detection	customer churn	facial recognition
assisting people with disabilities	customer experience	fitness tracking
auto-insurance risk prediction	customer retention	fraud detection
automated closed captioning	customer satisfaction	game playing
automated image captions	customer service	genomics and healthcare
automated investing	customer service agents	Geographic Information Systems (GIS)
autonomous ships	customized diets	GPS Systems
brain mapping	cybersecurity	health outcome improvement
caller identification	data mining	hospital readmission reduction
cancer diagnosis/treatment	data visualization	human genome sequencing
carbon emissions reduction	detecting new viruses	identity-theft prevention
classifying handwriting	diagnosing breast cancer	immunotherapy
computer vision	diagnosing heart disease	insurance pricing
credit scoring	diagnostic medicine	intelligent assistants
crime: predicting locations	disaster-victim identification	Internet of Things (IoT) and medical device monitoring
crime: predicting recidivism	drones	Internet of Things and weather forecasting
crime: predictive policing	dynamic driving routes	inventory control
crime: prevention	dynamic pricing	language translation
CRISPR gene editing	electronic health records	
crop-yield improvement	emotion detection	
	energy-consumption reduction	

71. Turck, M., and J. Hao, "Great Power, Great Responsibility: The 2018 Big Data & AI Landscape," `http://mattturck.com/bigdata2018/`.
72. Sawchik, T., *Big Data Baseball: Math, Miracles, and the End of a 20-Year Losing Streak* (New York, Flat Iron Books, 2015).
73. Ayres, I., *Super Crunchers* (Bantam Books, 2007), pp. 7–10.
74. Lewis, M., *Moneyball: The Art of Winning an Unfair Game* (W. W. Norton & Company, 2004).