Python® for Programmers

with introductory
AI case studies

- Natural Language Processing
- Data Mining Twitter®
- IBM® Watson™
- Machine Learning with scikit-learn®
- Deep Learning with Keras
- Big Data with Hadoop®
  Spark™, NoSQL and the Cloud
- Internet of Things (IoT)
- Python Standard Library
- Data Science Libraries:
  NumPy, Pandas, SciPy,
  NLTK, TextBlob, Tweepy,
  Matplotlib, Seaborn,
  Folium and more

Paul Deitel • Harvey Deitel
Python®
for Programmers

with introductory AI case studies

- Natural Language Processing
- Data Mining Twitter®
- IBM® Watson™
- Machine Learning with scikit-learn®
- Deep Learning with Keras
- Big Data with Hadoop®, Spark™, NoSQL and the Cloud
- Internet of Things (IoT)
- Python Standard Library
- Data Science Libraries: NumPy, Pandas, SciPy, NLTK, TextBlob, Tweepy, Matplotlib, Seaborn, Folium and more

PAUL DEITEL • HARVEY DEITEL
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
This page intentionally left blank
Contents

Preface xvii
Before You Begin xxxiii

I Introduction to Computers and Python 1
1.1 Introduction 2
1.2 A Quick Review of Object Technology Basics 3
1.3 Python 5
1.4 It’s the Libraries! 7
   1.4.1 Python Standard Library 7
   1.4.2 Data-Science Libraries 8
1.5 Test-Drives: Using IPython and Jupyter Notebooks 9
   1.5.1 Using IPython Interactive Mode as a Calculator 9
   1.5.2 Executing a Python Program Using the IPython Interpreter 10
   1.5.3 Writing and Executing Code in a Jupyter Notebook 12
1.6 The Cloud and the Internet of Things 16
   1.6.1 The Cloud 16
   1.6.2 Internet of Things 17
1.7 How Big Is Big Data? 17
   1.7.1 Big Data Analytics 22
   1.7.2 Data Science and Big Data Are Making a Difference: Use Cases 23
1.8 Case Study—A Big-Data Mobile Application 24
1.9 Intro to Data Science: Artificial Intelligence—at the Intersection of CS and Data Science 26
1.10 Wrap-Up 29

2 Introduction to Python Programming 31
2.1 Introduction 32
2.2 Variables and Assignment Statements 32
2.3 Arithmetic 33
2.4 Function print and an Intro to Single- and Double-Quoted Strings 36
2.5 Triple-Quoted Strings 38
2.6 Getting Input from the User 39
2.7 Decision Making: The if Statement and Comparison Operators 41
2.8 Objects and Dynamic Typing 45
2.9 Intro to Data Science: Basic Descriptive Statistics 46
2.10 Wrap-Up 48
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Control Statements</td>
<td>3.1 Introduction</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2 Control Statements</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.3 if Statement</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.4 if…else and if…elif…else Statements</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.5 while Statement</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.6 for Statement</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.6.1 Iterables, Lists and Iterators</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.6.2 Built-In range Function</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.7 Augmented Assignments</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.8 Sequence-Controlled Iteration; Formatted Strings</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.9 Sentinel-Controlled Iteration</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.10 Built-In Function range: A Deeper Look</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.11 Using Type Decimal for Monetary Amounts</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.12 break and continue Statements</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.13 Boolean Operators and, or and not</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.14 Intro to Data Science: Measures of Central Tendency—Mean, Median and Mode</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.15 Wrap-Up</td>
<td>69</td>
</tr>
<tr>
<td>4</td>
<td>Functions</td>
<td>4.1 Introduction</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.2 Defining Functions</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.3 Functions with Multiple Parameters</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.4 Random-Number Generation</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.5 Case Study: A Game of Chance</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.6 Python Standard Library</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.7 math Module Functions</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.8 Using IPython Tab Completion for Discovery</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.9 Default Parameter Values</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.10 Keyword Arguments</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.11 Arbitrary Argument Lists</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.12 Methods: Functions That Belong to Objects</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.13 Scope Rules</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.14 import: A Deeper Look</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.15 Passing Arguments to Functions: A Deeper Look</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.16 Recursion</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.17 Functional-Style Programming</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.18 Intro to Data Science: Measures of Dispersion</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.19 Wrap-Up</td>
<td>98</td>
</tr>
<tr>
<td>5</td>
<td>Sequences: Lists and Tuples</td>
<td>5.1 Introduction</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.2 Lists</td>
<td>102</td>
</tr>
</tbody>
</table>
5.3 Tuples  106
5.4 Unpacking Sequences  108
5.5 Sequence Slicing  110
5.6 del Statement  112
5.7 Passing Lists to Functions  113
5.8 Sorting Lists  115
5.9 Searching Sequences  116
5.10 Other List Methods  117
5.11 Simulating Stacks with Lists  119
5.12 List Comprehensions  120
5.13 Generator Expressions  121
5.14 Filter, Map and Reduce  122
5.15 Other Sequence Processing Functions  124
5.16 Two-Dimensional Lists  126
5.17 Intro to Data Science: Simulation and Static Visualizations  128
5.17.1 Sample Graphs for 600, 60,000 and 6,000,000 Die Rolls  128
5.17.2 Visualizing Die-Roll Frequencies and Percentages  129
5.18 Wrap-Up  135

6 Dictionaries and Sets  137
6.1 Introduction  138
6.2 Dictionaries
   6.2.1 Creating a Dictionary  138
   6.2.2 Iterating through a Dictionary  139
   6.2.3 Basic Dictionary Operations  140
   6.2.4 Dictionary Methods keys and values  141
   6.2.5 Dictionary Comparisons  143
   6.2.6 Example: Dictionary of Student Grades  143
   6.2.7 Example: Word Counts  144
   6.2.8 Dictionary Method update  146
   6.2.9 Dictionary Comprehensions  146
6.3 Sets
   6.3.1 Comparing Sets  148
   6.3.2 Mathematical Set Operations  150
   6.3.3 Mutable Set Operators and Methods  151
   6.3.4 Set Comprehensions  152
6.4 Intro to Data Science: Dynamic Visualizations
   6.4.1 How Dynamic Visualization Works  153
   6.4.2 Implementing a Dynamic Visualization  155
6.5 Wrap-Up  158

7 Array-Oriented Programming with NumPy  159
7.1 Introduction  160
7.2 Creating arrays from Existing Data  160
7.3 array Attributes  161
8 Strings: A Deeper Look 191
8.1 Introduction 192
8.2 Formatting Strings 193
  8.2.1 Presentation Types 193
  8.2.2 Field Widths and Alignment 194
  8.2.3 Numeric Formatting 195
  8.2.4 String's format Method 195
8.3 Concatenating and Repeating Strings 196
8.4 Stripping Whitespace from Strings 197
8.5 Changing Character Case 197
8.6 Comparison Operators for Strings 198
8.7 Searching for Substrings 198
8.8 Replacing Substrings 199
8.9 Splitting and Joining Strings 200
8.10 Characters and Character-Testing Methods 202
8.11 Raw Strings 203
8.12 Introduction to Regular Expressions 203
  8.12.1 re Module and Function fullmatch 204
  8.12.2 Replacing Substrings and Splitting Strings 207
  8.12.3 Other Search Functions; Accessing Matches 208
8.13 Intro to Data Science: Pandas, Regular Expressions and Data Munging 210
8.14 Wrap-Up 214

9 Files and Exceptions 217
9.1 Introduction 218
9.2 Files 219
9.3 Text-File Processing 219
  9.3.1 Writing to a Text File: Introducing the with Statement 220
  9.3.2 Reading Data from a Text File 221
9.4 Updating Text Files 222
9.5 Serialization with JSON 223
9.6 Focus on Security: pickle Serialization and Deserialization 226
9.7 Additional Notes Regarding Files 226
9.8 Handling Exceptions 227
9.8.1 Division by Zero and Invalid Input 227
9.8.2 try Statements 228
9.8.3 Catching Multiple Exceptions in One except Clause 230
9.8.4 What Exceptions Does a Function or Method Raise? 230
9.8.5 What Code Should Be Placed in a try Suite? 230
9.9 finally Clause 231
9.10 Explicitly Raising an Exception 233
9.11 (Optional) Stack Unwinding and Tracebacks 233
9.12 Intro to Data Science: Working with CSV Files 235
9.12.1 Python Standard Library Module csv 235
9.12.2 Reading CSV Files into Pandas DataFrames 237
9.12.3 Reading the Titanic Disaster Dataset 238
9.12.4 Simple Data Analysis with the Titanic Disaster Dataset 239
9.12.5 Passenger Age Histogram 240
9.13 Wrap-Up 241

10 Object-Oriented Programming 243
10.1 Introduction 244
10.2 Custom Class Account 246
10.2.1 Test-Driving Class Account 246
10.2.2 Account Class Definition 248
10.2.3 Composition: Object References as Members of Classes 249
10.3 Controlling Access to Attributes 249
10.4 Properties for Data Access 250
10.4.1 Test-Driving Class Time 250
10.4.2 Class Time Definition 252
10.4.3 Class Time Definition Design Notes 255
10.5 Simulating “Private” Attributes 256
10.6 Case Study: Card Shuffling and Dealing Simulation 258
10.6.1 Test-Driving Classes Card and DeckOfCards 258
10.6.2 Class Card—Introducing Class Attributes 259
10.6.3 Class DeckOfCards 261
10.6.4 Displaying Card Images with Matplotlib 263
10.7 Inheritance: Base Classes and Subclasses 266
10.8 Building an Inheritance Hierarchy; Introducing Polymorphism 267
10.8.1 Base Class CommissionEmployee 268
10.8.2 Subclass SalariedCommissionEmployee 270
10.8.3 Processing CommissionEmployees and SalariedCommissionEmployees Polymorphically 274
10.8.4 A Note About Object-Based and Object-Oriented Programming 274
10.9 Duck Typing and Polymorphism 275
10.10 Operator Overloading 276
  10.10.1 Test-Driving Class Complex 277
  10.10.2 Class Complex Definition 278
10.11 Exception Class Hierarchy and Custom Exceptions 279
10.12 Named Tuples 280
10.13 A Brief Intro to Python 3.7’s New Data Classes 281
  10.13.1 Creating a Card Data Class 282
  10.13.2 Using the Card Data Class 284
  10.13.3 Data Class Advantages over Named Tuples 286
  10.13.4 Data Class Advantages over Traditional Classes 286
10.14 Unit Testing with Docstrings and doctest 287
10.15 Namespaces and Scopes 290
10.16 Intro to Data Science: Time Series and Simple Linear Regression 293
10.17 Wrap-Up 301

II Natural Language Processing (NLP) 303
11.1 Introduction 304
11.2 TextBlob 305
  11.2.1 Create a TextBlob 307
  11.2.2 Tokenizing Text into Sentences and Words 307
  11.2.3 Parts-of-Speech Tagging 307
  11.2.4 Extracting Noun Phrases 308
  11.2.5 Sentiment Analysis with TextBlob’s Default Sentiment Analyzer 309
  11.2.6 Sentiment Analysis with the NaïveBayesAnalyzer 310
  11.2.7 Language Detection and Translation 311
  11.2.8 Inflection: Pluralization and Singularization 312
  11.2.9 Spell Checking and Correction 313
  11.2.10 Normalization: Stemming and Lemmatization 314
  11.2.11 Word Frequencies 314
  11.2.12 Getting Definitions, Synonyms and Antonyms from WordNet 315
  11.2.13 Deleting Stop Words 317
  11.2.14 n-grams 318
11.3 Visualizing Word Frequencies with Bar Charts and Word Clouds 319
  11.3.1 Visualizing Word Frequencies with Pandas 319
  11.3.2 Visualizing Word Frequencies with Word Clouds 321
11.4 Readability Assessment with Textatistic 324
11.5 Named Entity Recognition with spaCy 326
11.6 Similarity Detection with spaCy 327
11.7 Other NLP Libraries and Tools 328
11.8 Machine Learning and Deep Learning Natural Language Applications 328
11.9 Natural Language Datasets 329
11.10 Wrap-Up 330
## 12 Data Mining Twitter

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.1</td>
<td>Introduction</td>
<td>332</td>
</tr>
<tr>
<td>12.2</td>
<td>Overview of the Twitter APIs</td>
<td>334</td>
</tr>
<tr>
<td>12.3</td>
<td>Creating a Twitter Account</td>
<td>335</td>
</tr>
<tr>
<td>12.4</td>
<td>Getting Twitter Credentials—Creating an App</td>
<td>335</td>
</tr>
<tr>
<td>12.5</td>
<td>What’s in a Tweet?</td>
<td>337</td>
</tr>
<tr>
<td>12.6</td>
<td>Tweepy</td>
<td>340</td>
</tr>
<tr>
<td>12.7</td>
<td>Authenticating with Twitter Via Tweepy</td>
<td>341</td>
</tr>
<tr>
<td>12.8</td>
<td>Getting Information About a Twitter Account</td>
<td>342</td>
</tr>
<tr>
<td>12.9</td>
<td>Introduction to Tweepy Cursors: Getting an Account’s Followers and Friends</td>
<td>344</td>
</tr>
<tr>
<td>12.9.1</td>
<td>Determining an Account’s Followers</td>
<td>344</td>
</tr>
<tr>
<td>12.9.2</td>
<td>Determining Whom an Account Follows</td>
<td>346</td>
</tr>
<tr>
<td>12.9.3</td>
<td>Getting a User’s Recent Tweets</td>
<td>346</td>
</tr>
<tr>
<td>12.10</td>
<td>Searching Recent Tweets</td>
<td>347</td>
</tr>
<tr>
<td>12.11</td>
<td>Spotting Trends: Twitter Trends API</td>
<td>349</td>
</tr>
<tr>
<td>12.11.1</td>
<td>Places with Trending Topics</td>
<td>350</td>
</tr>
<tr>
<td>12.11.2</td>
<td>Getting a List of Trending Topics</td>
<td>351</td>
</tr>
<tr>
<td>12.11.3</td>
<td>Create a Word Cloud from Trending Topics</td>
<td>352</td>
</tr>
<tr>
<td>12.12</td>
<td>Cleaning/Preprocessing Tweets for Analysis</td>
<td>353</td>
</tr>
<tr>
<td>12.13</td>
<td>Twitter Streaming API</td>
<td>354</td>
</tr>
<tr>
<td>12.13.1</td>
<td>Creating a Subclass of StreamListener</td>
<td>355</td>
</tr>
<tr>
<td>12.13.2</td>
<td>Initiating Stream Processing</td>
<td>357</td>
</tr>
<tr>
<td>12.14</td>
<td>Tweet Sentiment Analysis</td>
<td>359</td>
</tr>
<tr>
<td>12.15</td>
<td>Geocoding and Mapping</td>
<td>362</td>
</tr>
<tr>
<td>12.15.1</td>
<td>Getting and Mapping the Tweets</td>
<td>364</td>
</tr>
<tr>
<td>12.15.2</td>
<td>Utility Functions in tweetutilities.py</td>
<td>367</td>
</tr>
<tr>
<td>12.15.3</td>
<td>Class LocationListener</td>
<td>369</td>
</tr>
<tr>
<td>12.16</td>
<td>Ways to Store Tweets</td>
<td>370</td>
</tr>
<tr>
<td>12.17</td>
<td>Twitter and Time Series</td>
<td>370</td>
</tr>
<tr>
<td>12.18</td>
<td>Wrap-Up</td>
<td>371</td>
</tr>
</tbody>
</table>

## 13 IBM Watson and Cognitive Computing

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.1</td>
<td>Introduction: IBM Watson and Cognitive Computing</td>
<td>374</td>
</tr>
<tr>
<td>13.2</td>
<td>IBM Cloud Account and Cloud Console</td>
<td>375</td>
</tr>
<tr>
<td>13.3</td>
<td>Watson Services</td>
<td>376</td>
</tr>
<tr>
<td>13.4</td>
<td>Additional Services and Tools</td>
<td>379</td>
</tr>
<tr>
<td>13.5</td>
<td>Watson Developer Cloud Python SDK</td>
<td>381</td>
</tr>
<tr>
<td>13.6</td>
<td>Case Study: Traveler’s Companion Translation App</td>
<td>381</td>
</tr>
<tr>
<td>13.6.1</td>
<td>Before You Run the App</td>
<td>382</td>
</tr>
<tr>
<td>13.6.2</td>
<td>Test-Driving the App</td>
<td>383</td>
</tr>
<tr>
<td>13.6.3</td>
<td>SimpleLanguageTranslator.py Script Walkthrough</td>
<td>384</td>
</tr>
<tr>
<td>13.7</td>
<td>Watson Resources</td>
<td>394</td>
</tr>
<tr>
<td>13.8</td>
<td>Wrap-Up</td>
<td>395</td>
</tr>
</tbody>
</table>
14 Machine Learning: Classification, Regression and Clustering

14.1 Introduction to Machine Learning
14.1.1 Scikit-Learn
14.1.2 Types of Machine Learning
14.1.3 Datasets Bundled with Scikit-Learn
14.1.4 Steps in a Typical Data Science Study

14.2 Case Study: Classification with k-Nearest Neighbors and the Digits Dataset, Part 1
14.2.1 k-Nearest Neighbors Algorithm
14.2.2 Loading the Dataset
14.2.3 Visualizing the Data
14.2.4 Splitting the Data for Training and Testing
14.2.5 Creating the Model
14.2.6 Training the Model
14.2.7 Predicting Digit Classes

14.3 Case Study: Classification with k-Nearest Neighbors and the Digits Dataset, Part 2
14.3.1 Metrics for Model Accuracy
14.3.2 K-Fold Cross-Validation
14.3.3 Running Multiple Models to Find the Best One
14.3.4 Hyperparameter Tuning

14.4 Case Study: Time Series and Simple Linear Regression

14.5 Case Study: Multiple Linear Regression with the California Housing Dataset
14.5.1 Loading the Dataset
14.5.2 Exploring the Data with Pandas
14.5.3 Visualizing the Features
14.5.4 Splitting the Data for Training and Testing
14.5.5 Training the Model
14.5.6 Testing the Model
14.5.7 Visualizing the Expected vs. Predicted Prices
14.5.8 Regression Model Metrics
14.5.9 Choosing the Best Model

14.6 Case Study: Unsupervised Machine Learning, Part 1—Dimensionality Reduction

14.7 Case Study: Unsupervised Machine Learning, Part 2—k-Means Clustering
14.7.1 Loading the Iris Dataset
14.7.2 Exploring the Iris Dataset: Descriptive Statistics with Pandas
14.7.3 Visualizing the Dataset with a Seaborn pairplot
14.7.4 Using a KMeans Estimator
14.7.5 Dimensionality Reduction with Principal Component Analysis
14.7.6 Choosing the Best Clustering Estimator

14.8 Wrap-Up
## 15 Deep Learning

15.1 Introduction  
15.1.1 Deep Learning Applications  
15.1.2 Deep Learning Demos  
15.1.3 Keras Resources  
15.2 Keras Built-In Datasets  
15.3 Custom Anaconda Environments  
15.4 Neural Networks  
15.5 Tensors  
15.6 Convolutional Neural Networks for Vision; Multi-Classification with the MNIST Dataset  
15.6.1 Loading the MNIST Dataset  
15.6.2 Data Exploration  
15.6.3 Data Preparation  
15.6.4 Creating the Neural Network  
15.6.5 Training and Evaluating the Model  
15.6.6 Saving and Loading a Model  
15.7 Visualizing Neural Network Training with TensorBoard  
15.8 ConvnetJS: Browser-Based Deep-Learning Training and Visualization  
15.9 Recurrent Neural Networks for Sequences; Sentiment Analysis with the IMDb Dataset  
15.9.1 Loading the IMDb Movie Reviews Dataset  
15.9.2 Data Exploration  
15.9.3 Data Preparation  
15.9.4 Creating the Neural Network  
15.9.5 Training and Evaluating the Model  
15.10 Tuning Deep Learning Models  
15.11 Convnet Models Pretrained on ImageNet  
15.12 Wrap-Up

## 16 Big Data: Hadoop, Spark, NoSQL and IoT

16.1 Introduction  
16.2 Relational Databases and Structured Query Language (SQL)  
16.2.1 A books Database  
16.2.2 SELECT Queries  
16.2.3 WHERE Clause  
16.2.4 ORDER BY Clause  
16.2.5 Merging Data from Multiple Tables: INNER JOIN  
16.2.6 INSERT INTO Statement  
16.2.7 UPDATE Statement  
16.2.8 DELETE FROM Statement  
16.3 NoSQL and NewSQL Big-Data Databases: A Brief Tour  
16.3.1 NoSQL Key–Value Databases  
16.3.2 NoSQL Document Databases
16.3.3 NoSQL Columnar Databases  518
16.3.4 NoSQL Graph Databases  519
16.3.5 NewSQL Databases  519
16.4 Case Study: A MongoDB JSON Document Database  520
  16.4.1 Creating the MongoDB Atlas Cluster  521
  16.4.2 Streaming Tweets into MongoDB  522
16.5 Hadoop  530
  16.5.1 Hadoop Overview  531
  16.5.2 Summarizing Word Lengths in Romeo and Juliet via MapReduce  533
  16.5.3 Creating an Apache Hadoop Cluster in Microsoft Azure HDInsight  533
  16.5.4 Hadoop Streaming  535
  16.5.5 Implementing the Mapper  536
  16.5.6 Implementing the Reducer  537
  16.5.7 Preparing to Run the MapReduce Example  537
  16.5.8 Running the MapReduce Job  538
16.6 Spark  541
  16.6.1 Spark Overview  541
  16.6.2 Docker and the Jupyter Docker Stacks  542
  16.6.3 Word Count with Spark  545
  16.6.4 Spark Word Count on Microsoft Azure  548
16.7 Spark Streaming: Counting Twitter Hashtags Using the pyspark-notebook Docker Stack  551
  16.7.1 Streaming Tweets to a Socket  551
  16.7.2 Summarizing Tweet Hashtags; Introducing Spark SQL  555
16.8 Internet of Things and Dashboards  560
  16.8.1 Publish and Subscribe  561
  16.8.2 Visualizing a PubNub Sample Live Stream with a Freeboard Dashboard  562
  16.8.3 Simulating an Internet-Connected Thermostat in Python  564
  16.8.4 Creating the Dashboard with Freeboard.io  566
  16.8.5 Creating a Python PubNub Subscriber  567
16.9 Wrap-Up  571

Index 573
Preface

“There’s gold in them thar hills!”

Welcome to Python for Programmers! In this book, you’ll learn hands-on with today’s most compelling, leading-edge computing technologies, and you’ll program in Python—one of the world’s most popular languages and the fastest growing among them.

Developers 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 a rapidly growing base of reusable software for an enormous range of application areas.

For many decades, some powerful trends have been in place. Computer hardware has rapidly been getting faster, cheaper and smaller. Internet bandwidth 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.

In computing today, the latest innovations are “all about the data”—data science, data analytics, big data, relational databases (SQL), and NoSQL and NewSQL databases, each of which we address along with an innovative treatment of Python programming.

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.” 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. 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.

1. Source unknown, frequently misattributed to Mark Twain.
Modular Architecture
The book’s modular architecture (please see the Table of Contents graphic on the book’s inside front cover) helps us meet the diverse needs of various professional audiences.

Chapters 1–10 cover Python programming. These chapters each include a 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 11–16, 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 case studies in Chapters 11–16. The “Chapter Dependencies” section of this Preface will help trainers plan their professional courses in the context of the book’s unique architecture.

Chapters 11–16 are loaded with cool, powerful, contemporary examples. They present hands-on implementation case studies on topics such as natural language processing, data mining Twitter, cognitive computing with IBM’s Watson, supervised machine learning with classification and regression, unsupervised machine learning with clustering, deep learning with convolutional neural networks, deep learning with recurrent neural networks, big data with Hadoop, Spark and NoSQL databases, the Internet of Things and more. Along the way, you’ll acquire a broad literacy of data science terms and concepts, ranging from brief definitions to using concepts in small, medium and large programs. Browsing the book’s detailed Table of Contents and Index will give you a sense of the breadth of coverage.

Key Features

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

- Keep it simple—In every aspect of the book, we strive for simplicity and clarity. For example, when we present natural language processing, we use the simple and intuitive TextBlob library rather than the more complex NLTK. In our deep learning presentation, we prefer Keras to TensorFlow. 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 also include 40 larger scripts and in-depth case studies.

- Keep it topical—We read scores of recent Python-programming and data science books, and 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.

Immediate-Feedback: Exploring, Discovering and Experimenting with IPython

- 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
Key Features

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. 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 approach, focusing on complete, working programs with live inputs and outputs. IPython’s “magic” is that it turns even snippets into code that “comes alive” as you enter each line. This promotes learning and encourages experimentation.

Python Programming Fundamentals

- First and foremost, this book provides rich Python coverage.
- We discuss Python’s programming models—procedural programming, functional-style programming and object-oriented programming.
- We use best practices, emphasizing current idiom.
- 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 most of them.

538 Code Examples

- You’ll get an engaging, challenging and entertaining introduction to Python with 538 real-world examples ranging from individual snippets to substantial computer science, data science, artificial intelligence and big data case studies.
- You’ll attack significant tasks 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, Textatistic, Tweepy, Scikit-learn, Keras), key visualization libraries (Matplotlib, Seaborn, Folium) and more.

Avoid Heavy Math in Favor of English Explanations

- We capture the conceptual essence of the mathematics and put it to work in our examples. We do this by using libraries such as statistics, NumPy, SciPy, pandas and many others, which hide the mathematical complexity. So, it’s straightforward for you 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.”

Visualizations

- 67 static, dynamic, animated and interactive visualizations (charts, graphs, pictures, animations etc.) help you understand concepts.
Rather than including a treatment of low-level graphics programming, 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.

Visualizations are 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—in big data, a visualization could be worth billions, trillions or even more items in a database. Visualizations enable you to “fly 40,000 feet above the data” to see it “in the large” and to get to know your data. Descriptive statistics help but can be misleading. For example, Anscombe’s quartet 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 you can conveniently customize the code and animation parameters, re-execute the animations and see the effects of the changes.

Data Experiences

Our Intro to Data Science sections and case studies in Chapters 11–16 provide rich data experiences.

You’ll work with many real-world datasets and data sources. There’s an enormous variety of free open datasets available online for you to experiment with. Some of the sites we reference list hundreds or thousands of datasets.

Many 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.

GitHub

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.

You’ll use an extraordinary range of free and open-source Python and data science libraries, and free, free-trial and freemium offerings of software and cloud services. Many of the libraries are hosted on GitHub.

**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.

- We encourage you to use free, free trial or freemium cloud services. We prefer those that don’t require a credit card because you don’t want 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. Evidence indicates that the speed of data creation is rapidly 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*.

- We include a 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. MongoDB is the most popular NoSQL database.

- We discuss **big data hardware and software infrastructure** in Chapter 16, “Big Data: Hadoop, Spark, NoSQL and IoT (Internet of Things).”

**Artificial Intelligence Case Studies**

- In case study Chapters 11–15, we present **artificial intelligence** topics, including natural language processing, data mining Twitter to perform sentiment analysis, cognitive computing with IBM Watson, supervised machine learning, unsupervised machine learning and deep learning. Chapter 16 presents the big data hardware and software infrastructure that enables computer scientists and data scientists to implement leading-edge AI-based solutions.

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

- There’s little reason today for most application developers to build *custom* data structures. The book features a rich **two-chapter treatment of Python’s built-in data structures**—lists, tuples, dictionaries and sets—with which most data-structuring tasks can be accomplished.

---


8. [https://analyticsweek.com/content/big-data-facts/](https://analyticsweek.com/content/big-data-facts/)
Array-Oriented Programming with NumPy Arrays and Pandas Series/DataFrames

- We also focus on three key data structures from open-source libraries—NumPy arrays, pandas Series and pandas DataFrames. These 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 used frequently in the data science chapters.
- 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

- We emphasize using the huge 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 apps, creating objects from existing classes (usually in one or two lines of code) and making them “jump, dance and sing.” This object-based programming enables you to build impressive applications quickly and concisely, which is a significant part of Python’s appeal.
- With this approach, you’ll be able to use machine learning, deep learning and other AI technologies to quickly solve a wide range of intriguing problems, including cognitive computing challenges like speech recognition and computer vision.

Object-Oriented Programming

- Developing custom classes is a crucial object-oriented programming skill, along with inheritance, polymorphism and duck typing. We discuss these in Chapter 10.
- Chapter 10 includes a discussion of unit testing with doctest and a fun card-shuffling-and-dealing simulation.
- Chapters 11–16 require only a few straightforward custom class definitions. In Python, you’ll probably use more of an object-based programming approach than full-out object-oriented programming.

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 discuss reproducibility throughout the book in the context of programming techniques and software such as Jupyter Notebooks and Docker.

Performance
- We use the `%timeit` profiling tool in several examples 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
- 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 applications 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 expeditiously.

Chapter Dependencies
If you’re a trainer planning your syllabus for a professional training course or a developer deciding which chapters to read, this section will help you make the best decisions. Please read the one-page color Table of Contents on the book’s inside front cover—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 11–16 requires only Chapters 1–5 and small portions of Chapters 6–10 as discussed below.

Part 1: Python Fundamentals Quickstart
We recommend that you read all the chapters in order:
- Chapter 1, *Introduction to Computers and Python*, introduces concepts that lay the groundwork for the Python programming in Chapters 2–10 and the big data, artificial-intelligence and cloud-based case studies in Chapters 11–16. The chapter also includes test-drives of the IPython interpreter and Jupyter Notebooks.
- Chapter 2, *Introduction to Python Programming*, presents Python programming fundamentals with code examples illustrating key language features.
- Chapter 3, *Control Statements*, presents Python’s control statements and introduces basic list processing.
- Chapter 4, *Functions*, introduces custom functions, 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 introducing functional-style programming.
Part 2: Python Data Structures, Strings and Files
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 in this chapter is not dependent on the chapter’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.

Part 3: Python High-End Topics
The following summarizes inter-chapter dependencies for Python Chapter 10 and assumes that you’ve read Chapters 1–5.

- Chapter 10, Object-Oriented Programming—The Intro to Data Science section requires pandas DataFrame features from Intro to Data Science Section 7.14. Trainers wanting to cover only classes and objects can present Sections 10.1–10.6. Trainers 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.

Part 4: AI, Cloud and Big Data Case Studies
The following summary of inter-chapter dependencies for Chapters 11–16 assumes that you’ve read Chapters 1–5. Most of Chapters 11–16 also require dictionary fundamentals from Section 6.2.

- Chapter 11, Natural Language Processing (NLP), uses pandas DataFrame features from Section 7.14’s Intro to Data Science.
- Chapter 12, 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 11.2) and Word clouds (Section 11.3). Several examples require defining a class via inheritance (Chapter 10).
- Chapter 13, IBM Watson and Cognitive Computing, uses built-in function open and the with state ment (Section 9.3).
- Chapter 14, 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 15, Deep Learning, requires NumPy fundamentals (Chapter 7), string method join (Section 8.9), general machine-learning concepts from
Chapter 14 and features from Chapter 14’s Case Study: Classification with k-Nearest Neighbors and the Digits Dataset.

- **Chapter 16, 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 11.2.13) and from Chapter 12, 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 you can simply mimic the class definitions we provide without reading Chapter 10.

**Jupyter Notebooks**

For your convenience, we provide the book’s code 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.

Jupyter Notebooks is a free, open-source project that enables you 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.9

In our experience, it’s a wonderful learning environment and **rapid prototyping tool**. For this reason, we use **Jupyter Notebooks** rather than a traditional 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 mechanisms10 (see “Getting Jupyter Help” later in this Preface). See the Before You Begin section that follows this Preface for software installation details and see the test-drives in Section 1.5 for information on running the book’s examples.

**Collaboration and Sharing Results**

Working in teams and communicating research results are both important for developers in or 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.\(^\text{11}\)
- The article “Language-Agnostic Reproducible Data Analysis Using Literate Programming,”\(^\text{12}\) 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).

Docker

In Chapter 16, we’ll use 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 16 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 other developers to recreate the environment you used, then reproduce your work, and will help you reproduce your own results. In Chapter 16, 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.

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.\(^\text{13}\) We include a significant hands-on Watson treatment. We use the free Watson Developer Cloud: Python SDK, which provides 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

---


\(^{12}\) http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0164023.

\(^{13}\) http://whatis.techtarget.com/definition/cognitive-computing.


Understanding, Personality Insights, Speech to Text, Text to Speech, Tone Analyzer and Visual Recognition.

**Watson’s Lite Tier Services and a Cool Watson Case Study**
IBM encourages learning and experimentation by providing free lite tiers for many of its APIs. In Chapter 13, you’ll try demos of many Watson services. 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!

**Teaching Approach**
*Python for Programmers* contains a rich collection of examples drawn from many fields. You’ll work through interesting, real-world examples using 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 refer to 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**
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

**538 Code Examples**
The book’s 538 examples contain approximately 4000 lines of code. This is a relatively small amount 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**
We include abundant tables, line drawings, and static, dynamic and interactive visualizations.

---

16. Always check the latest terms on IBM’s website, as the terms and services may change.
17. [https://console.bluemix.net/catalog/](https://console.bluemix.net/catalog/).
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.

Software Used in the Book

The software we use 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 the IPython interpreter, Jupyter Notebooks and Spyder, considered one of the best Python data science IDEs. We use only IPython and Jupyter Notebooks for program development in the book. The Before You Begin section following this Preface discusses installing Anaconda and a few 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 Language Reference:
  https://docs.python.org/3/reference/index.html
- The Python Standard Library:
  https://docs.python.org/3/library/index.html
- Python documentation list:
  https://docs.python.org/3/

Getting Your Questions Answered

Popular Python and general programming online forums include:

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

Also, many vendors provide forums for their tools and libraries. Many of the libraries you’ll use in this book are managed and maintained at github.com. Some library main-
tainers 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

**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

**Supplements**

To get the most out of the presentation, you should execute each code example in parallel with reading the corresponding discussion in the book. On the book’s web page at http://www.deitel.com we provide:

- Downloadable Python source code (.py files) and Jupyter Notebooks (.ipynb files) for the book’s code examples.
- Getting Started videos showing how to use the code examples with IPython and Jupyter Notebooks. We also introduce these tools in Section 1.5.
- Blog posts and book updates.

For download instructions, see the Before You Begin section that follows this Preface.

**Keeping in Touch with the Authors**

For answers to questions 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)

---

18. Our website is undergoing a major upgrade. If you do not find something you need, please write to us directly at deitel@deitel.com.
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 efforts and 25-year mentorship of our friend and colleague Mark L. Taub, Vice President of the Pearson IT Professional Group. Mark and his team publish our professional books, LiveLessons video products and Learning Paths in the Safari service (https://learning.oreilly.com/). They also sponsor our Safari live online training seminars. Julie Nahil managed the book’s production. We selected the cover art and Chuti Prasertsith designed the cover.

We wish to acknowledge the efforts of our reviewers. Patricia Byron-Kimball and Meghan Jacoby 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.

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

<table>
<thead>
<tr>
<th>Proposal Reviewers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr. Irene Bruno, Associate Professor in the Department of Information Sciences and Technology, George Mason University</td>
</tr>
<tr>
<td>Lance Bryant, Associate Professor, Department of Mathematics, Shippensburg University</td>
</tr>
<tr>
<td>Dr. Marsha Davis, Department Chair of Mathematical Sciences, Eastern Connecticut State University</td>
</tr>
<tr>
<td>Dr. Dr. Mark Pauley, Senior Research Fellow, Bioinformatics, School of Interdisciplinary Informatics, University of Nebraska at Omaha</td>
</tr>
<tr>
<td>Sean Raleigh, Associate Professor of Mathematics, Chair of Data Science, Westminster College</td>
</tr>
<tr>
<td>Alison Sanchez, Assistant Professor in Economics, University of San Diego</td>
</tr>
<tr>
<td>Dr. Harvey Siy, Associate Professor of Computer Science, Information Science and Technology, University of Nebraska at Omaha</td>
</tr>
<tr>
<td>Jamie Whitacre, Independent Data Science Consultant</td>
</tr>
</tbody>
</table>

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, data science, AI, big data and the cloud. We wish you great success!

Paul and Harvey Deitel

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.


To contact Deitel & Associates, Inc. and the authors, or to request a proposal on-site, instructor-led training, write to:

ditel@deitel.com
To learn more about Deitel on-site corporate training, visit
http://www.deitel.com/training

Individuals wishing to purchase Deitel books can do so at
https://www.amazon.com

Bulk orders by corporations, the government, the military and academic institutions
should be placed directly with Pearson. For more information, visit
Before You Begin

This section contains information you should review before using this book. We'll post updates at: http://www.deitel.com.

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 our *Python for Programmers* web page at:

http://www.deitel.com

Click the **Download Examples** link to save the file to your local computer. Most web browsers place the file in your user account’s `Downloads` folder. 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 1.5’s test drives.

The `examples` folder contains one subfolder per chapter. These are named `ch##`, where `##` is the two-digit chapter number 01 to 16—for example, `ch01`. Except for Chapters 13, 15 and 16, 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 13 contains one application. Chapters 15 and 16 explain where to find the files you need in the ch15 and ch16 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

You may want to analyze your Python code using the Prospector analysis tool, which checks your code for common errors and helps you improve it. To install Prospector and the Python libraries it uses, run the following command in the command-line window:

```
ip 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

---

\(^1\) \url{https://github.com/matplotlib/jupyter-matplotlib}.
Before You Begin

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 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.com²

² Our website is undergoing a major upgrade. If you do not find something you need, please write to us directly at deitel@deitel.com.

You’re now ready to begin reading Python for Programmers. We hope you enjoy the book!
Sequences: Lists and Tuples

Objectives
In this chapter, you’ll:
■ Create and initialize lists and tuples.
■ Refer to elements of lists, tuples and strings.
■ Sort and search lists, and search tuples.
■ Pass lists and tuples to functions and methods.
■ Use list methods to perform common manipulations, such as searching for items, sorting a list, inserting items and removing items.
■ Use additional Python functional-style programming capabilities, including lambdas and the functional-style programming operations filter, map and reduce.
■ Use functional-style list comprehensions to create lists quickly and easily, and use generator expressions to generate values on demand.
■ Use two-dimensional lists.
■ Enhance your analysis and presentation skills with the Seaborn and Matplotlib visualization libraries.
Chapter 5  Sequences: Lists and Tuples

5.1 Introduction

In the last two chapters, we briefly introduced the list and tuple sequence types for representing ordered collections of items. *Collections* are prepackaged data structures consisting of related data items. Examples of collections include your favorite songs on your smartphone, your contacts list, a library’s books, your cards in a card game, your favorite sports team’s players, the stocks in an investment portfolio, patients in a cancer study and a shopping list. Python’s built-in collections enable you to store and access data conveniently and efficiently. In this chapter, we discuss lists and tuples in more detail.

We’ll demonstrate common list and tuple manipulations. You’ll see that lists (which are modifiable) and tuples (which are not) have many common capabilities. Each can hold items of the same or different types. Lists can *dynamically resize* as necessary, growing and shrinking at execution time. We discuss one-dimensional and two-dimensional lists.

In the preceding chapter, we demonstrated random-number generation and simulated rolling a six-sided die. We conclude this chapter with our next Intro to Data Science section, which uses the visualization libraries Seaborn and Matplotlib to interactively develop static bar charts containing the die frequencies. In the next chapter’s Intro to Data Science section, we’ll present an animated visualization in which the bar chart changes *dynamically* as the number of die rolls increases—you’ll see the law of large numbers “in action.”

5.2 Lists

Here, we discuss lists in more detail and explain how to refer to particular list elements. Many of the capabilities shown in this section apply to all sequence types.

Creating a List

Lists typically store *homogeneous data*, that is, values of the *same* data type. Consider the list `c`, which contains five integer elements:

```
In [1]: c = [-45, 6, 0, 72, 1543]
```

```
In [2]: c
Out[2]: [-45, 6, 0, 72, 1543]
```
They also may store heterogeneous data, that is, data of many different types. For example, the following list contains a student’s first name (a string), last name (a string), grade point average (a float) and graduation year (an int):

['Mary', 'Smith', 3.57, 2022]

**Accessing Elements of a List**

You reference a list element by writing the list’s name followed by the element’s index (that is, its position number) enclosed in square brackets ([], known as the subscription operator). The following diagram shows the list c labeled with its element names:

![Diagram showing list element names and their positions]

The first element in a list has the index 0. So, in the five-element list c, the first element is named c[0] and the last is c[4]:

```
In [3]: c[0]
Out[3]: -45
```

```
In [4]: c[4]
Out[4]: 1543
```

**Determining a List’s Length**

To get a list’s length, use the built-in `len` function:

```
In [5]: len(c)
Out[5]: 5
```

**Accessing Elements from the End of the List with Negative Indices**

Lists also can be accessed from the end by using negative indices:

```
In [6]: c[-1]
Out[6]: 1543
```

```
In [7]: c[-5]
Out[7]: -45
```

**Indices Must Be Integers or Integer Expressions**

An index must be an integer or integer expression (or a slice, as we’ll soon see):
Using a non-integer index value causes a `TypeError`.

**Lists Are Mutable**
Lists are mutable—their elements can be modified:

```python
In [12]: c
Out[12]: [-45, 6, 0, 72, 17]
```

You'll soon see that you also can insert and delete elements, changing the list's length.

**Some Sequences Are Immutable**
Python's string and tuple sequences are immutable—they cannot be modified. You can get the individual characters in a string, but attempting to assign a new value to one of the characters causes a `TypeError`:

```python
In [13]: s = 'hello'
In [14]: s[0]
Out[14]: 'h'
In [15]: s[0] = 'H'
-------------------------------------------------------------------------
TypeError                               Traceback (most recent call last)
<ipython-input-15-812ef2514689> in <module>()
----> 1 s[0] = 'H'
TypeError: 'str' object does not support item assignment
```

**Attempting to Access a Nonexistent Element**
Using an out-of-range list, tuple or string index causes an `IndexError`:

```python
In [16]: c[100]
-------------------------------------------------------------------------
IndexError                              Traceback (most recent call last)
<ipython-input-16-9a31ea1e1a13> in <module>()
----> 1 c[100]
IndexError: list index out of range
```

**Using List Elements in Expressions**
List elements may be used as variables in expressions:

```python
In [17]: c[0] + c[1] + c[2]
Out[17]: -39
```

**Appending to a List with +=**
Let's start with an empty list `[]`, then use a `for` statement and `+=` to append the values 1 through 5 to the list—the list grows dynamically to accommodate each item:

```python
In [18]: a_list = []
```
When the left operand of `+=` is a list, the right operand must be an *iterable*; otherwise, a `TypeError` occurs. In snippet [19]'s suite, the square brackets around `number` create a one-element list, which we append to `a_list`. If the right operand contains multiple elements, `+=` appends them all. The following appends the characters of 'Python' to the list `letters`:

```python
In [21]: letters = []
In [22]: letters += 'Python'
In [23]: letters
Out[23]: ['P', 'y', 't', 'h', 'o', 'n']
```

If the right operand of `+=` is a tuple, its elements also are appended to the list. Later in the chapter, we’ll use the list method `append` to add items to a list.

**Concatenating Lists with +**

You can concatenate two lists, two tuples or two strings using the + operator. The result is a *new* sequence of the same type containing the left operand’s elements followed by the right operand’s elements. The original sequences are unchanged:

```python
In [24]: list1 = [10, 20, 30]
In [25]: list2 = [40, 50]
In [26]: concatenated_list = list1 + list2
In [27]: concatenated_list
Out[27]: [10, 20, 30, 40, 50]
```

A `TypeError` occurs if the + operator’s operands are different sequence types—for example, concatenating a list and a tuple is an error.

**Using for and range to Access List Indices and Values**

List elements also can be accessed via their indices and the subscription operator (`[]`):

```python
In [28]: for i in range(len(concatenated_list)):
    ...:     print(f'{i}: {concatenated_list[i]}')
    ...
0: 10
1: 20
2: 30
3: 40
4: 50
```

The function call `range(len(concatenated_list))` produces a sequence of integers representing `concatenated_list`’s indices (in this case, 0 through 4). When looping in this manner, you must ensure that indices remain in range. Soon, we’ll show a safer way to access element indices and values using built-in function `enumerate`. 
Comparison Operators
You can compare entire lists element-by-element using comparison operators:

```python
In [29]: a = [1, 2, 3]
In [30]: b = [1, 2, 3]
In [31]: c = [1, 2, 3, 4]
In [32]: a == b  # True: corresponding elements in both are equal
Out[32]: True
In [33]: a == c  # False: a and c have different elements and lengths
Out[33]: False
In [34]: a < c  # True: a has fewer elements than c
Out[34]: True
In [35]: c >= b  # True: elements 0-2 are equal but c has more elements
Out[35]: True
```

5.3 Tuples
As discussed in the preceding chapter, tuples are immutable and typically store heterogeneous data, but the data can be homogeneous. A tuple's length is its number of elements and cannot change during program execution.

Creating Tuples
To create an empty tuple, use empty parentheses:

```python
In [1]: student_tuple = ()
In [2]: student_tuple
Out[2]: ()
In [3]: len(student_tuple)
Out[3]: 0
```

Recall that you can pack a tuple by separating its values with commas:

```python
In [4]: student_tuple = 'John', 'Green', 3.3
In [5]: student_tuple
Out[5]: ('John', 'Green', 3.3)
In [6]: len(student_tuple)
Out[6]: 3
```

When you output a tuple, Python always displays its contents in parentheses. You may surround a tuple's comma-separated list of values with optional parentheses:

```python
In [7]: another_student_tuple = ('Mary', 'Red', 3.3)
In [8]: another_student_tuple
Out[8]: ('Mary', 'Red', 3.3)
```
The following code creates a one-element tuple:

```
In [9]: a_singleton_tuple = ('red',)  # note the comma
```

```
In [10]: a_singleton_tuple
Out[10]: ('red',)
```

The comma (,) that follows the string 'red' identifies `a_singleton_tuple` as a tuple—the parentheses are optional. If the comma were omitted, the parentheses would be redundant, and `a_singleton_tuple` would simply refer to the string 'red' rather than a tuple.

### Accessing Tuple Elements
A tuple’s elements, though related, are often of multiple types. Usually, you do not iterate over them. Rather, you access each individually. Like list indices, tuple indices start at 0. The following code creates `time_tuple` representing an hour, minute and second, displays the tuple, then uses its elements to calculate the number of seconds since midnight—note that we perform a different operation with each value in the tuple:

```
In [11]: time_tuple = (9, 16, 1)
```

```
In [12]: time_tuple
Out[12]: (9, 16, 1)
```

```
In [13]: time_tuple[0] * 3600 + time_tuple[1] * 60 + time_tuple[2]
Out[13]: 33361
```

Assigning a value to a tuple element causes a `TypeError`.

### Adding Items to a String or Tuple
As with lists, the `+=` augmented assignment statement can be used with strings and tuples, even though they’re immutable. In the following code, after the two assignments, `tuple1` and `tuple2` refer to the same tuple object:

```
In [14]: tuple1 = (10, 20, 30)
```

```
In [15]: tuple2 = tuple1
```

```
In [16]: tuple2
Out[16]: (10, 20, 30)
```

```
In [17]: tuple1 += (40, 50)
```

```
In [18]: tuple1
Out[18]: (10, 20, 30, 40, 50)
```

```
In [19]: tuple2
Out[19]: (10, 20, 30)
```

For a string or tuple, the item to the right of `+=` must be a string or tuple, respectively—mixing types causes a `TypeError`.

**Chapter 5  Sequences: Lists and Tuples**

### Appending Tuples to Lists

You can use `+=` to append a tuple to a list:

```python
In [20]: numbers = [1, 2, 3, 4, 5]
In [21]: numbers += (6, 7)
In [22]: numbers
Out[22]: [1, 2, 3, 4, 5, 6, 7]
```

### Tuples May Contain Mutable Objects

Let's create a `student_tuple` with a first name, last name and list of grades:

```python
In [23]: student_tuple = ('Amanda', 'Blue', [98, 75, 87])
```

Even though the tuple is immutable, its list element is mutable:

```python
In [24]: student_tuple[2][1] = 85
In [25]: student_tuple
Out[25]: ('Amanda', 'Blue', [98, 85, 87])
```

In the *double-subscripted name* `student_tuple[2][1]`, Python views `student_tuple[2]` as the element of the tuple containing the list `[98, 75, 87]`, then uses `[1]` to access the list element containing 75. The assignment in snippet [24] replaces that grade with 85.

### 5.4 Unpacking Sequences

The previous chapter introduced tuple unpacking. You can unpack any sequence's elements by assigning the sequence to a comma-separated list of variables. A ValueError occurs if the number of variables to the left of the assignment symbol is not identical to the number of elements in the sequence on the right:

```python
In [1]: student_tuple = ('Amanda', [98, 85, 87])
In [2]: first_name, grades = student_tuple
In [3]: first_name
Out[3]: 'Amanda'
In [4]: grades
Out[4]: [98, 85, 87]
```

The following code unpacks a string, a list and a sequence produced by `range`:

```python
In [5]: first, second = 'hi'
In [6]: print(f'{first} {second}')
hi

In [7]: number1, number2, number3 = [2, 3, 5]
In [8]: print(f'{number1} {number2} {number3}')
2 3 5
In [9]: number1, number2, number3 = range(10, 40, 10)
In [10]: print(f'{number1} {number2} {number3}')
10 20 30
```
5.4 Unpacking Sequences

Swapping Values Via Packing and Unpacking
You can swap two variables’ values using sequence packing and unpacking:

```python
In [11]: number1 = 99
In [12]: number2 = 22
In [13]: number1, number2 = (number2, number1)
In [14]: print(f'number1 = {number1}; number2 = {number2}')
number1 = 22; number2 = 99
```

Accessing Indices and Values Safely with Built-in Function `enumerate`
Earlier, we called `range` to produce a sequence of index values, then accessed list elements in a for loop using the index values and the subscription operator (`[]`). This is error-prone because you could pass the wrong arguments to `range`. If any value produced by `range` is an out-of-bounds index, using it as an index causes an `IndexError`.

The preferred mechanism for accessing an element’s index and value is the built-in function `enumerate`. This function receives an iterable and creates an iterator that, for each element, returns a tuple containing the element’s index and value. The following code uses the built-in function `list` to create a list containing `enumerate`’s results:

```python
In [15]: colors = ['red', 'orange', 'yellow']
In [16]: list(enumerate(colors))
Out[16]: [(0, 'red'), (1, 'orange'), (2, 'yellow')]
```

Similarly the built-in function `tuple` creates a tuple from a sequence:

```python
In [17]: tuple(enumerate(colors))
Out[17]: ((0, 'red'), (1, 'orange'), (2, 'yellow'))
```

The following for loop unpacks each tuple returned by `enumerate` into the variables `index` and `value` and displays them:

```python
In [18]: for index, value in enumerate(colors):
   ...:     print(f'{index}: {value}')
   ...
0: red
1: orange
2: yellow
```

Creating a Primitive Bar Chart
The following script creates a primitive bar chart where each bar’s length is made of asterisks (*) and is proportional to the list’s corresponding element value. We use the function `enumerate` to access the list’s indices and values safely. To run this example, change to this chapter’s ch05 examples folder, then enter:

```
ipython fig05_01.py
```
or, if you’re in IPython already, use the command:

```
run fig05_01.py
```
Chapter 5  Sequences: Lists and Tuples

The for statement uses enumerate to get each element’s index and value, then displays a formatted line containing the index, the element value and the corresponding bar of asterisks. The expression

```python
"*" * value
```

creates a string consisting of value asterisks. When used with a sequence, the multiplication operator (*) repeats the sequence—in this case, the string "*"—value times. Later in this chapter, we’ll use the open-source Seaborn and Matplotlib libraries to display a publication-quality bar chart visualization.

5.5 Sequence Slicing

You can slice sequences to create new sequences of the same type containing subsets of the original elements. Slice operations can modify mutable sequences—those that do not modify a sequence work identically for lists, tuples and strings.

**Specifying a Slice with Starting and Ending Indices**

Let’s create a slice consisting of the elements at indices 2 through 5 of a list:

```python
In [1]: numbers = [2, 3, 5, 7, 11, 13, 17, 19]
In [2]: numbers[2:6]
Out[2]: [5, 7, 11, 13]
```

The slice copies elements from the starting index to the left of the colon (2) up to, but not including, the ending index to the right of the colon (6). The original list is not modified.

**Specifying a Slice with Only an Ending Index**

If you omit the starting index, 0 is assumed. So, the slice `numbers[:6]` is equivalent to the slice `numbers[0:6]`:

```python
In [3]: numbers[:6]
Out[3]: [2, 3, 5, 7, 11, 13]
```
5.5  Sequence Slicing

Specifying a Slice with Only a Starting Index
If you omit the ending index, Python assumes the sequence’s length (8 here), so snippet [5]’s slice contains the elements of numbers at indices 6 and 7:

```
In [5]: numbers[6:]
Out[5]: [17, 19]
```

Specifying a Slice with No Indices
Omitting both the start and end indices copies the entire sequence:

```
In [7]: numbers[:]
Out[7]: [2, 3, 5, 7, 11, 13, 17, 19]
```

Though slices create new objects, slices make shallow copies of the elements—that is, they copy the elements’ references but not the objects they point to. So, in the snippet above, the new list’s elements refer to the same objects as the original list’s elements, rather than to separate copies. In the “Array-Oriented Programming with NumPy” chapter, we’ll explain deep copying, which actually copies the referenced objects themselves, and we’ll point out when deep copying is preferred.

Slicing with Steps
The following code uses a step of 2 to create a slice with every other element of numbers:

```
In [8]: numbers[::2]
Out[8]: [2, 5, 11, 17]
```

We omitted the start and end indices, so 0 and len(numbers) are assumed, respectively.

Slicing with Negative Indices and Steps
You can use a negative step to select slices in reverse order. The following code concisely creates a new list in reverse order:

```
In [9]: numbers[::-1]
Out[9]: [19, 17, 13, 11, 7, 5, 3, 2]
```

This is equivalent to:

```
In [10]: numbers[-1:-9:-1]
Out[10]: [19, 17, 13, 11, 7, 5, 3, 2]
```

Modifying Lists Via Slices
You can modify a list by assigning to a slice of it—the rest of the list is unchanged. The following code replaces numbers’ first three elements, leaving the rest unchanged:

```
In [11]: numbers[0:3] = ['two', 'three', 'five']

In [12]: numbers
Out[12]: ['two', 'three', 'five', 7, 11, 13, 17, 19]
```
Chapter 5  Sequences: Lists and Tuples

The following deletes only the first three elements of *numbers* by assigning an *empty* list to the three-element slice:

```
In [13]: numbers[0:3] = []
```

```
In [14]: numbers
Out[14]: [7, 11, 13, 17, 19]
```

The following assigns a list's elements to a slice of every other element of *numbers*:

```
In [15]: numbers = [2, 3, 5, 7, 11, 13, 17, 19]
```

```
In [16]: numbers[::2] = [100, 100, 100, 100]
```

```
In [17]: numbers
Out[17]: [100, 3, 100, 7, 100, 13, 100, 19]
```

```
In [18]: id(numbers)
Out[18]: 4434456648
```

Let's delete all the elements in *numbers*, leaving the *existing* list empty:

```
In [19]: numbers[:] = []
```

```
In [20]: numbers
Out[20]: []
```

```
In [21]: id(numbers)
Out[21]: 4434456648
```

Deleting *numbers*’ contents (snippet [19]) is different from assigning *numbers* a *new* empty list [] (snippet [22]). To prove this, we display *numbers*’ identity after each operation. The identities are different, so they represent separate objects in memory:

```
In [22]: numbers = []
```

```
In [23]: numbers
Out[23]: []
```

```
In [24]: id(numbers)
Out[24]: 4406030920
```

When you assign a new object to a variable (as in snippet [21]), the original object will be garbage collected if no other variables refer to it.

### 5.6 del Statement

The *del* statement also can be used to remove elements from a list and to delete variables from the interactive session. You can remove the element at any valid index or the element(s) from any valid slice.

**Deleting the Element at a Specific List Index**

Let's create a list, then use *del* to remove its last element:

```
In [1]: numbers = list(range(0, 10))
```

```
In [2]: numbers
Out[2]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```
Deleting a Slice from a List
The following deletes the list’s first two elements:

```python
In [5]: del numbers[0:2]
In [6]: numbers
Out[6]: [2, 3, 4, 5, 6, 7, 8]
```

The following uses a step in the slice to delete every other element from the entire list:

```python
In [7]: del numbers[::2]
In [8]: numbers
Out[8]: [3, 5, 7]
```

Deleting a Slice Representing the Entire List
The following code deletes all of the list’s elements:

```python
In [9]: del numbers[:]
In [10]: numbers
Out[10]: []
```

Deleting a Variable from the Current Session
The `del` statement can delete any variable. Let’s delete `numbers` from the interactive session, then attempt to display the variable’s value, causing a `NameError`:

```python
In [11]: del numbers
```

```python
NameError: name 'numbers' is not defined
```

5.7 Passing Lists to Functions
In the last chapter, we mentioned that all objects are passed by reference and demonstrated passing an immutable object as a function argument. Here, we discuss references further by examining what happens when a program passes a mutable list object to a function.

**Passing an Entire List to a Function**
Consider the function `modify_elements`, which receives a reference to a list and multiplies each of the list’s element values by 2:

```python
In [3]: del numbers[-1]
In [4]: numbers
Out[4]: [0, 1, 2, 3, 4, 5, 6, 7, 8]
```
Function `modify_elements`' `items` parameter receives a reference to the original list, so the statement in the loop's suite modifies each element in the original list object.

### Passing a Tuple to a Function

When you pass a tuple to a function, attempting to modify the tuple's immutable elements results in a `TypeError`:

```
In [5]: numbers_tuple = (10, 20, 30)
In [6]: numbers_tuple
Out[6]: (10, 20, 30)
In [7]: modify_elements(numbers_tuple)
```

```
  ----> 1 modify_elements(numbers_tuple)
<ipython-input-1-27acb8f8f44c> in modify_elements(items)
     2     """"""Multiplies all element values in items by 2."""
     3     for i in range(len(items)):
     ----> 4         items[i] *= 2
     5
     6
```

```
TypeError: 'tuple' object does not support item assignment
```

Recall that tuples may contain mutable objects, such as lists. Those objects still can be modified when a tuple is passed to a function.

### A Note Regarding Tracebacks

The previous traceback shows the two snippets that led to the `TypeError`. The first is snippet [7]'s function call. The second is snippet [1]'s function definition. Line numbers precede each snippet's code. We've demonstrated mostly single-line snippets. When an exception occurs in such a snippet, it's always preceded by `----> 1`, indicating that line 1 (the snippet's only line) caused the exception. Multiline snippets like the definition of `modify_elements` show consecutive line numbers starting at 1. The notation `----> 4` above indicates that the exception occurred in line 4 of `modify_elements`. No matter how long the traceback is, the last line of code with `---->` caused the exception.
5.8 Sorting Lists

Sorting enables you to arrange data either in ascending or descending order.

**Sorting a List in Ascending Order**

List method `sort` modifies a list to arrange its elements in ascending order:

```python
In [1]: numbers = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]
In [2]: numbers.sort()
In [3]: numbers
Out[3]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

**Sorting a List in Descending Order**

To sort a list in descending order, call list method `sort` with the optional keyword argument `reverse` set to True (False is the default):

```python
In [4]: numbers.sort(reverse=True)
In [5]: numbers
Out[5]: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

**Built-In Function `sorted`**

Built-in function `sorted` returns a new list containing the sorted elements of its argument, the original sequence is unmodified. The following code demonstrates function `sorted` for a list, a string and a tuple:

```python
In [6]: numbers = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]
In [7]: ascending_numbers = sorted(numbers)
In [8]: ascending_numbers
Out[8]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
In [9]: numbers
Out[9]: [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]
In [10]: letters = 'fadgchjebi'
In [11]: ascending_letters = sorted(letters)
In [12]: ascending_letters
Out[12]: ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
In [13]: letters
Out[13]: 'fadgchjebi'
In [14]: colors = ('red', 'orange', 'yellow', 'green', 'blue')
In [15]: ascending_colors = sorted(colors)
In [16]: ascending_colors
Out[16]: ['blue', 'green', 'orange', 'red', 'yellow']
In [17]: colors
Out[17]: ('red', 'orange', 'yellow', 'green', 'blue')
Use the optional keyword argument reverse with the value True to sort the elements in descending order.

## 5.9 Searching Sequences

Often, you’ll want to determine whether a sequence (such as a list, tuple or string) contains a value that matches a particular *key* value. *Searching* is the process of locating a key.

### List Method `index`

List method `index` takes as an argument a search key—the value to locate in the list—then searches through the list from index 0 and returns the index of the *first* element that matches the search key:

```python
In [1]: numbers = [3, 7, 1, 4, 2, 8, 5, 6]

In [2]: numbers.index(5)
Out[2]: 6
```

A `ValueError` occurs if the value you’re searching for is not in the list.

### Specifying the Starting Index of a Search

Using method `index`’s optional arguments, you can search a subset of a list’s elements. You can use `*=` to *multiply a sequence*—that is, append a sequence to itself multiple times. After the following snippet, `numbers` contains two copies of the original list’s contents:

```python
In [3]: numbers *= 2

In [4]: numbers
Out[4]: [3, 7, 1, 4, 2, 8, 5, 6, 3, 7, 1, 4, 2, 8, 5, 6]
```

The following code searches the updated list for the value 5 starting from index 7 and continuing through the end of the list:

```python
In [5]: numbers.index(5, 7)
Out[5]: 14
```

### Specifying the Starting and Ending Indices of a Search

Specifying the starting and ending indices causes `index` to search from the starting index up to but not including the ending index location. The call to `index` in snippet [5]:

```python
numbers.index(5, 7)
```

assumes the length of `numbers` as its optional third argument and is equivalent to:

```python
numbers.index(5, 7, len(numbers))
```

The following looks for the value 7 in the range of elements with indices 0 through 3:

```python
In [6]: numbers.index(7, 0, 4)
Out[6]: 1
```

### Operators `in` and `not in`

Operator `in` tests whether its right operand’s iterable contains the left operand’s value:

```python
In [7]: 1000 in numbers
Out[7]: False
```
Similarly, operator \texttt{not in} tests whether its right operand’s iterable does \textit{not} contain the left operand’s value:

\begin{verbatim}
In [9]: 1000 not in numbers
Out[9]: True
In [10]: 5 not in numbers
Out[10]: False
\end{verbatim}

\textbf{Using Operator \texttt{in} to Prevent a \texttt{ValueError}}

You can use the operator \texttt{in} to ensure that calls to method \texttt{index} do not result in \texttt{ValueError}s for search keys that are not in the corresponding sequence:

\begin{verbatim}
In [11]: key = 1000
In [12]: if key in numbers:
...:     print(f'\texttt{{found \{key\} at index {numbers.index(search_key)}}}')
...: else:
...:     print(f'\texttt{{\{key\} not found}}')
...:
1000 not found
\end{verbatim}

\textbf{Built-In Functions \texttt{any} and \texttt{all}}

Sometimes you simply need to know whether \textit{any} item in an iterable is \texttt{True} or whether \textit{all} the items are \texttt{True}. The built-in function \texttt{any} returns \texttt{True} if any item in its iterable argument is \texttt{True}. The built-in function \texttt{all} returns \texttt{True} if all items in its iterable argument are \texttt{True}. Recall that nonzero values are \texttt{True} and \texttt{0} is \texttt{False}. Non-empty iterable objects also evaluate to \texttt{True}, whereas any empty iterable evaluates to \texttt{False}. Functions \texttt{any} and \texttt{all} are additional examples of internal iteration in functional-style programming.

\section*{5.10 Other List Methods}

Lists also have methods that add and remove elements. Consider the list \texttt{color_names}:

\begin{verbatim}
In [1]: color_names = ['orange', 'yellow', 'green']
\end{verbatim}

\textbf{Inserting an Element at a Specific List Index}

Method \texttt{insert} adds a new item at a specified index. The following inserts \texttt{'red'} at index 0:

\begin{verbatim}
In [2]: color_names.insert(0, 'red')
In [3]: color_names
Out[3]: ['red', 'orange', 'yellow', 'green']
\end{verbatim}

\textbf{Adding an Element to the End of a List}

You can add a new item to the end of a list with method \texttt{append}:

\begin{verbatim}
In [4]: color_names.append('blue')
In [5]: color_names
Out[5]: ['red', 'orange', 'yellow', 'green', 'blue']
\end{verbatim}
Adding All the Elements of a Sequence to the End of a List
Use list method `extend` to add all the elements of another sequence to the end of a list:

```python
In [6]: color_names.extend(['indigo', 'violet'])
```

```python
In [7]: color_names
Out[7]: ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']
```

This is the equivalent of using `+=`. The following code adds all the characters of a string then all the elements of a tuple to a list:

```python
In [8]: sample_list = []
In [9]: s = 'abc'
In [10]: sample_list.extend(s)
In [11]: sample_list
Out[11]: ['a', 'b', 'c']
In [12]: t = (1, 2, 3)
In [13]: sample_list.extend(t)
In [14]: sample_list
Out[14]: ['a', 'b', 'c', 1, 2, 3]
```

Rather than creating a temporary variable, like `t`, to store a tuple before appending it to a list, you might want to pass a tuple directly to `extend`. In this case, the tuple's parentheses are required, because `extend` expects one iterable argument:

```python
In [15]: sample_list.extend((4, 5, 6))  # note the extra parentheses
```

```python
In [16]: sample_list
Out[16]: ['a', 'b', 'c', 1, 2, 3, 4, 5, 6]
```

A `TypeError` occurs if you omit the required parentheses.

Removing the First Occurrence of an Element in a List
Method `remove` deletes the first element with a specified value—a `ValueError` occurs if `remove`'s argument is not in the list:

```python
In [17]: color_names.remove('green')
```

```python
In [18]: color_names
Out[18]: ['red', 'orange', 'yellow', 'blue', 'indigo', 'violet']
```

Emptying a List
To delete all the elements in a list, call method `clear`:

```python
In [19]: color_names.clear()
```

```python
In [20]: color_names
Out[20]: []
```

This is the equivalent of the previously shown slice assignment

```
color_names[:] = []
```
Counting the Number of Occurrences of an Item
List method `count` searches for its argument and returns the number of times it is found:

```python
In [21]: responses = [1, 2, 5, 4, 3, 5, 2, 1, 3, 3,
...:                   1, 4, 3, 3, 3, 2, 3, 3, 2, 2]
...:
In [22]: for i in range(1, 6):
...:     print(f'{i} appears {responses.count(i)} times in responses')
...:
1 appears 3 times in responses
2 appears 5 times in responses
3 appears 8 times in responses
4 appears 2 times in responses
5 appears 2 times in responses
```

Reversing a List’s Elements
List method `reverse` reverses the contents of a list in place, rather than creating a reversed copy, as we did with a slice previously:

```python
In [23]: color_names = ['red', 'orange', 'yellow', 'green', 'blue']
In [24]: color_names.reverse()
In [25]: color_names
Out[25]: ['blue', 'green', 'yellow', 'orange', 'red']
```

Copying a List
List method `copy` returns a new list containing a shallow copy of the original list:

```python
In [26]: copied_list = color_names.copy()
In [27]: copied_list
Out[27]: ['blue', 'green', 'yellow', 'orange', 'red']
```

This is equivalent to the previously demonstrated slice operation:
```
copied_list = color_names[:]
```

5.1.1 Simulating Stacks with Lists
The preceding chapter introduced the function-call stack. Python does not have a built-in stack type, but you can think of a stack as a constrained list. You `push` using list method `append`, which adds a new element to the end of the list. You `pop` using list method `pop` with no arguments, which removes and returns the item at the end of the list.

Let’s create an empty list called `stack`, push (append) two strings onto it, then pop the strings to confirm they’re retrieved in last-in, first-out (LIFO) order:

```python
In [1]: stack = []
In [2]: stack.append('red')
In [3]: stack
Out[3]: ['red']
In [4]: stack.append('green')
```
For each `pop` snippet, the value that `pop` removes and returns is displayed. Popping from an empty stack causes an `IndexError`, just like accessing a nonexistent list element with `[]`. To prevent an `IndexError`, ensure that `len(stack)` is greater than 0 before calling `pop`. You can run out of memory if you keep pushing items faster than you pop them.

You also can use a list to simulate another popular collection called a queue in which you insert at the back and delete from the front. Items are retrieved from queues in first-in, first-out (FIFO) order.

### 5.12 List Comprehensions

Here, we continue discussing functional-style features with list comprehensions—a concise and convenient notation for creating new lists. List comprehensions can replace many for statements that iterate over existing sequences and create new lists, such as:

```python
In [1]: list1 = []
In [2]: for item in range(1, 6):
   ...:     list1.append(item)
   ...:
In [3]: list1
Out[3]: [1, 2, 3, 4, 5]
```

**Using a List Comprehension to Create a List of Integers**

We can accomplish the same task in a single line of code with a list comprehension:

```python
In [4]: list2 = [item for item in range(1, 6)]
In [5]: list2
Out[5]: [1, 2, 3, 4, 5]
```
Like snippet [2]’s for statement, the list comprehension’s for clause

```python
for item in range(1, 6)
```

iterates over the sequence produced by `range(1, 6)`. For each `item`, the list comprehension evaluates the expression to the left of the for clause and places the expression’s value (in this case, the `item` itself) in the new list. Snippet [4]’s particular comprehension could have been expressed more concisely using the function `list`:

```python
list2 = list(range(1, 6))
```

### Mapping: Performing Operations in a List Comprehension’s Expression
A list comprehension’s expression can perform tasks, such as calculations, that map elements to new values (possibly of different types). Mapping is a common functional-style programming operation that produces a result with the same number of elements as the original data being mapped. The following comprehension maps each value to its cube with the expression `item ** 3`:

```python
In [6]: list3 = [item ** 3 for item in range(1, 6)]
In [7]: list3
Out[7]: [1, 8, 27, 64, 125]
```

### Filtering: List Comprehensions with if Clauses
Another common functional-style programming operation is filtering elements to select only those that match a condition. This typically produces a list with fewer elements than the data being filtered. To do this in a list comprehension, use the if clause. The following includes in `list4` only the even values produced by the for clause:

```python
In [8]: list4 = [item for item in range(1, 11) if item % 2 == 0]
In [9]: list4
Out[9]: [2, 4, 6, 8, 10]
```

### List Comprehension That Processes Another List’s Elements
The for clause can process any iterable. Let’s create a list of lowercase strings and use a list comprehension to create a new list containing their uppercase versions:

```python
In [10]: colors = ['red', 'orange', 'yellow', 'green', 'blue']
In [11]: colors2 = [item.upper() for item in colors]
In [12]: colors2
Out[12]: ['RED', 'ORANGE', 'YELLOW', 'GREEN', 'BLUE']
In [13]: colors
Out[13]: ['red', 'orange', 'yellow', 'green', 'blue']
```

### 5.13 Generator Expressions
A generator expression is similar to a list comprehension, but creates an iterable generator object that produces values on demand. This is known as lazy evaluation. List comprehensions use greedy evaluation—they create lists immediately when you execute them. For large numbers of items, creating a list can take substantial memory and time. So generator
expressions can reduce your program’s memory consumption and improve performance if the whole list is not needed at once.

Generator expressions have the same capabilities as list comprehensions, but you define them in parentheses instead of square brackets. The generator expression in snippet [2] squares and returns only the odd values in numbers:

To show that a generator expression does not create a list, let’s assign the preceding snippet’s generator expression to a variable and evaluate the variable:

The text “generator object <genexpr>” indicates that square_of_odds is a generator object that was created from a generator expression (genexpr).

**5.14 Filter, Map and Reduce**

The preceding section introduced several functional-style features—list comprehensions, filtering and mapping. Here we demonstrate the built-in `filter` and `map` functions for filtering and mapping, respectively. We continue discussing reductions in which you process a collection of elements into a single value, such as their count, total, product, average, minimum or maximum.

**Filtering a Sequence’s Values with the Built-In `filter` Function**

Let’s use built-in function `filter` to obtain the odd values in numbers:

Like data, Python functions are objects that you can assign to variables, pass to other functions and return from functions. Functions that receive other functions as arguments are a functional-style capability called higher-order functions. For example, `filter`’s first argument must be a function that receives one argument and returns `True` if the value should be included in the result. The function `is_odd` returns `True` if its argument is odd. The `filter` function calls `is_odd` once for each value in its second argument’s iterable (numbers). Higher-order functions may also return a function as a result.
Function `filter` returns an iterator, so `filter`'s results are not produced until you iterate through them. This is another example of lazy evaluation. In snippet [3], function `list` iterates through the results and creates a list containing them. We can obtain the same results as above by using a list comprehension with an if clause:

```python
In [4]: [item for item in numbers if is_odd(item)]
Out[4]: [3, 7, 1, 9, 5]
```

### Using a lambda Rather than a Function
For simple functions like `is_odd` that return only a single expression's value, you can use a lambda expression (or simply a lambda) to define the function inline where it’s needed—typically as it’s passed to another function:

```python
In [5]: list(filter(lambda x: x % 2 != 0, numbers))
Out[5]: [3, 7, 1, 9, 5]
```

We pass `filter`'s return value (an iterator) to function `list` here to convert the results to a list and display them.

A lambda expression is an anonymous function—that is, a function without a name. In the filter call

```python
filter(lambda x: x % 2 != 0, numbers)
```

the first argument is the lambda

```python
lambda x: x % 2 != 0
```

A lambda begins with the `lambda` keyword followed by a comma-separated parameter list, a colon (:) and an expression. In this case, the parameter list has one parameter named `x`. A lambda *implicitly* returns its expression’s value. So any simple function of the form

```python
def function_name(parameter_list):
    return expression
```

may be expressed as a more concise lambda of the form

```python
lambda parameter_list: expression
```

### Mapping a Sequence’s Values to New Values
Let’s use built-in function `map` with a lambda to square each value in `numbers`:

```python
In [6]: numbers
Out[6]: [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]
In [7]: list(map(lambda x: x ** 2, numbers))
Out[7]: [100, 9, 49, 1, 81, 16, 4, 64, 25, 36]
```

Function `map`'s first argument is a function that receives one value and returns a new value—in this case, a lambda that squares its argument. The second argument is an iterable of values to map. Function `map` uses lazy evaluation. So, we pass to the `list` function the iterator that `map` returns. This enables us to iterate through and create a list of the mapped values. Here’s an equivalent list comprehension:

```python
In [8]: [item ** 2 for item in numbers]
Out[8]: [100, 9, 49, 1, 81, 16, 4, 64, 25, 36]
```
Combining filter and map
You can combine the preceding filter and map operations as follows:

\[
\begin{align*}
\text{In [9]: } & \text{list(map(lambda x: x ** 2,}
\text{\quad \cdots: \quad filter(lambda x: x % 2 != 0, \text{numbers})}}) \\
\text{\quad \cdots: \quad Out[9]: } & [9, \text{ } 49, 1, \text{ } 81, \text{ } 25]
\end{align*}
\]

There is a lot going on in snippet [9], so let’s take a closer look at it. First, filter returns an iterable representing only the odd values of numbers. Then map returns an iterable representing the squares of the filtered values. Finally, list uses map’s iterable to create the list. You might prefer the following list comprehension to the preceding snippet:

\[
\begin{align*}
\text{In [10]: } & [x ** 2 \text{ for } x \text{ in numbers } \text{ if } x \% 2 \neq 0] \\
\text{Out[10]: } & [9, \text{ } 49, 1, \text{ } 81, \text{ } 25]
\end{align*}
\]

For each value of \(x\) in numbers, the expression \(x ** 2\) is performed only if the condition \(x \% 2 = 0\) is True.

Reduction: Totaling the Elements of a Sequence with sum
As you know reductions process a sequence’s elements into a single value. You’ve performed reductions with the built-in functions len, sum, min and max. You also can create custom reductions using the functools module’s reduce function. See https://docs.python.org/3/library/functools.html for a code example. When we investigate big data and Hadoop in Chapter 16, we’ll demonstrate MapReduce programming, which is based on the filter, map and reduce operations in functional-style programming.

5.15 Other Sequence Processing Functions
Python provides other built-in functions for manipulating sequences.

Finding the Minimum and Maximum Values Using a Key Function
We’ve previously shown the built-in reduction functions min and max using arguments, such as ints or lists of ints. Sometimes you’ll need to find the minimum and maximum of more complex objects, such as strings. Consider the following comparison:

\[
\begin{align*}
\text{In [1]: } & \text{ 'Red' } < \text{ 'orange'} \\
\text{Out[1]: } & \text{ True}
\end{align*}
\]

The letter 'R' “comes after” 'o' in the alphabet, so you might expect 'Red' to be less than 'orange' and the condition above to be False. However, strings are compared by their characters’ underlying numerical values, and lowercase letters have higher numerical values than uppercase letters. You can confirm this with built-in function ord, which returns the numerical value of a character:

\[
\begin{align*}
\text{In [2]: } & \text{ ord('R')} \\
\text{Out[2]: } & 82
\end{align*}
\]

\[
\begin{align*}
\text{In [3]: } & \text{ ord('o')} \\
\text{Out[3]: } & 111
\end{align*}
\]

Consider the list colors, which contains strings with uppercase and lowercase letters:

\[
\begin{align*}
\text{In [4]: } & \text{ colors } = \text{ ['Red', 'orange', 'Yellow', 'green', 'Blue']}
\end{align*}
\]
Let’s assume that we’d like to determine the minimum and maximum strings using alphabetical order, not numerical (lexicographical) order. If we arrange colors alphabetically

'Blue', 'green', 'orange', 'Red', 'Yellow'

you can see that 'Blue' is the minimum (that is, closest to the beginning of the alphabet), and 'Yellow' is the maximum (that is, closest to the end of the alphabet).

Since Python compares strings using numerical values, you must first convert each string to all lowercase or all uppercase letters. Then their numerical values will also represent alphabetical ordering. The following snippets enable min and max to determine the minimum and maximum strings alphabetically:

```
In [5]: min(colors, key=lambda s: s.lower())
Out[5]: 'Blue'

In [6]: max(colors, key=lambda s: s.lower())
Out[6]: 'Yellow'
```

The key keyword argument must be a one-parameter function that returns a value. In this case, it’s a lambda that calls string method lower to get a string’s lowercase version. Functions min and max call the key argument’s function for each element and use the results to compare the elements.

**Iterating Backward Through a Sequence**

Built-in function reversed returns an iterator that enables you to iterate over a sequence’s values backward. The following list comprehension creates a new list containing the squares of numbers’ values in reverse order:

```
In [7]: numbers = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]

In [7]: reversed_numbers = [item for item in reversed(numbers)]

In [8]: reversed_numbers
Out[8]: [36, 25, 64, 4, 16, 81, 1, 49, 9, 100]
```

**Combining Iterables into Tuples of Corresponding Elements**

Built-in function zip enables you to iterate over multiple iterables of data at the same time. The function receives as arguments any number of iterables and returns an iterator that produces tuples containing the elements at the same index in each. For example, snippet [11]’s call to zip produces the tuples ('Bob', 3.5), ('Sue', 4.0) and ('Amanda', 3.75) consisting of the elements at index 0, 1 and 2 of each list, respectively:

```
In [9]: names = ['Bob', 'Sue', 'Amanda']

In [10]: grade_point_averages = [3.5, 4.0, 3.75]

In [11]: for name, gpa in zip(names, grade_point_averages):
               ...:     print(f'Name={name}; GPA={gpa}')
               ...
Name=Bob; GPA=3.5
Name=Sue; GPA=4.0
Name=Amanda; GPA=3.75
```

We unpack each tuple into name and gpa and display them. Function zip’s shortest argument determines the number of tuples produced. Here both have the same length.
5.16 Two-Dimensional Lists

Lists can contain other lists as elements. A typical use of such nested (or multidimensional) lists is to represent tables of values consisting of information arranged in rows and columns. To identify a particular table element, we specify two indices—by convention, the first identifies the element’s row, the second the element’s column.

Lists that require two indices to identify an element are called two-dimensional lists (or double-indexed lists or double-subscripted lists). Multidimensional lists can have more than two indices. Here, we introduce two-dimensional lists.

Creating a Two-Dimensional List

Consider a two-dimensional list with three rows and four columns (i.e., a 3-by-4 list) that might represent the grades of three students who each took four exams in a course:

In [1]: a = [[77, 68, 86, 73], [96, 87, 89, 81], [70, 90, 86, 81]]

Writing the list as follows makes its row and column tabular structure clearer:

```python
a = [[77, 68, 86, 73],  # first student’s grades
     [96, 87, 89, 81],  # second student’s grades
     [70, 90, 86, 81]]  # third student’s grades
```

Illustrating a Two-Dimensional List

The diagram below shows the list `a`, with its rows and columns of exam grade values:

<table>
<thead>
<tr>
<th>Row 0</th>
<th>Row 1</th>
<th>Row 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>77</td>
<td>96</td>
<td>70</td>
</tr>
<tr>
<td>68</td>
<td>87</td>
<td>90</td>
</tr>
<tr>
<td>86</td>
<td>89</td>
<td>86</td>
</tr>
<tr>
<td>73</td>
<td>81</td>
<td>81</td>
</tr>
</tbody>
</table>

Identifying the Elements in a Two-Dimensional List

The following diagram shows the names of list `a`’s elements:

Every element is identified by a name of the form `a[i][j]`—`a` is the list’s name, and `i` and `j` are the indices that uniquely identify each element’s row and column, respectively. The element names in row 0 all have 0 as the first index. The element names in column 3 all have 3 as the second index.
In the two-dimensional list \(a\):

- \(77, 68, 86\) and \(73\) initialize \(a[0][0]\), \(a[0][1]\), \(a[0][2]\) and \(a[0][3]\), respectively,
- \(96, 87, 89\) and \(81\) initialize \(a[1][0]\), \(a[1][1]\), \(a[1][2]\) and \(a[1][3]\), respectively, and
- \(70, 90, 86\) and \(81\) initialize \(a[2][0]\), \(a[2][1]\), \(a[2][2]\) and \(a[2][3]\), respectively.

A list with \(m\) rows and \(n\) columns is called an \(m\text{-by-}n\) list and has \(m \times n\) elements.

The following nested for statement outputs the rows of the preceding two-dimensional list one row at a time:

```python
In [2]: for row in a:
    ...:     for item in row:
    ...:         print(item, end=' ')  
    ...:     print()
    ...
77 68 86 73
96 87 89 81
70 90 86 81
```

**How the Nested Loops Execute**

Let’s modify the nested loop to display the list’s name and the row and column indices and value of each element:

```python
In [3]: for i, row in enumerate(a):
    ...:     for j, item in enumerate(row):
    ...:         print(f'a[{i}][{j}]=item ', end=' ')  
    ...:     print()
    ...
    a[0][0]=77  a[0][1]=68  a[0][2]=86  a[0][3]=73
    a[1][0]=96  a[1][1]=87  a[1][2]=89  a[1][3]=81
```

The outer for statement iterates over the two-dimensional list’s rows one row at a time. During each iteration of the outer for statement, the inner for statement iterates over each column in the current row. So in the first iteration of the outer loop, row 0 is

\([77, 68, 86, 73]\)

and the nested loop iterates through this list’s four elements \(a[0][0]=77\), \(a[0][1]=68\), \(a[0][2]=86\) and \(a[0][3]=73\).

In the second iteration of the outer loop, row 1 is

\([96, 87, 89, 81]\)

and the nested loop iterates through this list’s four elements \(a[1][0]=96\), \(a[1][1]=87\), \(a[1][2]=89\) and \(a[1][3]=81\).

In the third iteration of the outer loop, row 2 is

\([70, 90, 86, 81]\)

and the nested loop iterates through this list’s four elements \(a[2][0]=70\), \(a[2][1]=90\), \(a[2][2]=86\) and \(a[2][3]=81\).

In the “Array-Oriented Programming with NumPy” chapter, we’ll cover the NumPy library’s ndarray collection and the Pandas library’s DataFrame collection. These enable
you to manipulate multidimensional collections more concisely and conveniently than the two-dimensional list manipulations you’ve seen in this section.

5.17 Intro to Data Science: Simulation and Static Visualizations

The last few chapters’ Intro to Data Science sections discussed basic descriptive statistics. Here, we focus on visualizations, which help you “get to know” your data. Visualizations give you a powerful way to understand data that goes beyond simply looking at raw data.

We use two open-source visualization libraries—Seaborn and Matplotlib—to display static bar charts showing the final results of a six-sided-die-rolling simulation. The Seaborn visualization library is built over the Matplotlib visualization library and simplifies many Matplotlib operations. We’ll use aspects of both libraries, because some of the Seaborn operations return objects from the Matplotlib library. In the next chapter’s Intro to Data Science section, we’ll make things “come alive” with dynamic visualizations.

5.17.1 Sample Graphs for 600, 60,000 and 6,000,000 Die Rolls

The screen capture below shows a vertical bar chart that for 600 die rolls summarizes the frequencies with which each of the six faces appear, and their percentages of the total. Seaborn refers to this type of graph as a bar plot:

Here we expect about 100 occurrences of each die face. However, with such a small number of rolls, none of the frequencies is exactly 100 (though several are close) and most of the percentages are not close to 16.667% (about \( \frac{1}{6} \))th. As we run the simulation for 60,000 die rolls, the bars will become much closer in size. At 6,000,000 die rolls, they’ll appear to be exactly the same size. This is the “law of large numbers” at work. The next chapter will show the lengths of the bars changing dynamically.

We’ll discuss how to control the plot’s appearance and contents, including:

- the graph title inside the window (Rolling a Six-Sided Die 600 Times),
- the descriptive labels Die Value for the x-axis and Frequency for the y-axis,
5.17 Intro to Data Science: Simulation and Static Visualizations

- the text displayed above each bar, representing the frequency and percentage of the total rolls, and
- the bar colors.

We’ll use various Seaborn default options. For example, Seaborn determines the text labels along the x-axis from the die face values 1–6 and the text labels along the y-axis from the actual die frequencies. Behind the scenes, Matplotlib determines the positions and sizes of the bars, based on the window size and the magnitudes of the values the bars represent. It also positions the Frequency axis’s numeric labels based on the actual die frequencies that the bars represent. There are many more features you can customize. You should tweak these attributes to your personal preferences.

The first screen capture below shows the results for 60,000 die rolls—imagine trying to do this by hand. In this case, we expect about 10,000 of each face. The second screen capture below shows the results for 6,000,000 rolls—surely something you’d never do by hand! In this case, we expect about 1,000,000 of each face, and the frequency bars appear to be identical in length (they’re close but not exactly the same length). Note that with more die rolls, the frequency percentages are much closer to the expected 16.667%.

5.17.2 Visualizing Die-Roll Frequencies and Percentages

In this section, you’ll interactively develop the bar plots shown in the preceding section.

Launching IPython for Interactive Matplotlib Development

IPython has built-in support for interactively developing Matplotlib graphs, which you also need to develop Seaborn graphs. Simply launch IPython with the command:

```
ipython --matplotlib
```

Importing the Libraries

First, let’s import the libraries we’ll use:

```
In [1]: import matplotlib.pyplot as plt

In [2]: import numpy as np
```
Chapter 5  Sequences: Lists and Tuples

1. The matplotlib.pyplot module contains the Matplotlib library's graphing capabilities that we use. This module typically is imported with the name plt.

2. The NumPy (Numerical Python) library includes the function unique that we’ll use to summarize the die rolls. The numpy module typically is imported as np.

3. The random module contains Python’s random-number generation functions.

4. The seaborn module contains the Seaborn library’s graphing capabilities we use. This module typically is imported with the name sns. Search for why this curious abbreviation was chosen.

Rolling the Die and Calculating Die Frequencies
Next, let’s use a list comprehension to create a list of 600 random die values, then use NumPy’s unique function to determine the unique roll values (most likely all six possible face values) and their frequencies:

```python
In [5]: rolls = [random.randrange(1, 7) for i in range(600)]

In [6]: values, frequencies = np.unique(rolls, return_counts=True)
```

The NumPy library provides the high-performance ndarray collection, which is typically much faster than lists. Though we do not use ndarray directly here, the NumPy unique function expects an ndarray argument and returns an ndarray. If you pass a list (like rolls), NumPy converts it to an ndarray for better performance. The ndarray that unique returns we’ll simply assign to a variable for use by a Seaborn plotting function.

Specifying the keyword argument return_counts=True tells unique to count each unique value’s number of occurrences. In this case, unique returns a tuple of two one-dimensional ndarrays containing the sorted unique values and the corresponding frequencies, respectively. We unpack the tuple’s ndarrays into the variables values and frequencies. If return_counts is False, only the list of unique values is returned.

Creating the Initial Bar Plot
Let’s create the bar plot’s title, set its style, then graph the die faces and frequencies:

```python
In [7]: title = f'Rolling a Six-Sided Die {len(rolls):,} Times'

In [8]: sns.set_style('whitegrid')

In [9]: axes = sns.barplot(x=values, y=frequencies, palette='bright')
```

Snippet [7]’s f-string includes the number of die rolls in the bar plot’s title. The comma (,) format specifier in

```python
{len(rolls):,}
```

displays the number with thousands separators—so, 60000 would be displayed as 60,000.

By default, Seaborn plots graphs on a plain white background, but it provides several styles to choose from (‘darkgrid’, ‘whitegrid’, ‘dark’, ‘white’ and ‘ticks’). Snippet

---

1. We’ll run a performance comparison in Chapter 7 where we discuss ndarray in depth.
specifies the 'whitegrid' style, which displays light-gray horizontal lines in the vertical bar plot. These help you see more easily how each bar’s height corresponds to the numeric frequency labels at the bar plot’s left side.

Snippet [9] graphs the die frequencies using Seaborn’s `barplot` function. When you execute this snippet, the following window appears (because you launched IPython with the `--matplotlib` option):

Seaborn interacts with Matplotlib to display the bars by creating a Matplotlib `Axes` object, which manages the content that appears in the window. Behind the scenes, Seaborn uses a Matplotlib `Figure` object to manage the window in which the `Axes` will appear. Function barplot’s first two arguments are ndarrays containing the x-axis and y-axis values, respectively. We used the optional `palette` keyword argument to choose Seaborn’s predefined color palette 'bright'. You can view the palette options at:

https://seaborn.pydata.org/tutorial/color_palettes.html

Function barplot returns the `Axes` object that it configured. We assign this to the variable `axes` so we can use it to configure other aspects of our final plot. Any changes you make to the bar plot after this point will appear immediately when you execute the corresponding snippet.

**Setting the Window Title and Labeling the x- and y-Axes**

The next two snippets add some descriptive text to the bar plot:

```
In [10]: axes.set_title(title)
Out[10]: Text(0.5,1,'Rolling a Six-Sided Die 600 Times')
```

```
In [11]: axes.set(xlabel='Die Value', ylabel='Frequency')
Out[11]: [Text(92.6667,0.5,'Frequency'), Text(0.5,58.7667,'Die Value')]
```

Snippet [10] uses the axes object’s `set_title` method to display the title string centered above the plot. This method returns a Text object containing the title and its location in the window, which IPython simply displays as output for confirmation. You can ignore the Out[]s in the snippets above.

Snippet [11] add labels to each axis. The `set` method receives keyword arguments for the Axes object’s properties to set. The method displays the xlabel text along the x-axis,
and the ylabel text along the y-axis, and returns a list of Text objects containing the labels and their locations. The bar plot now appears as follows:

![Bar Plot](image)

**Finalizing the Bar Plot**

The next two snippets complete the graph by making room for the text above each bar, then displaying it:

```python
In [12]: axes.set_ylim(top=max(frequencies) * 1.10)
Out[12]: (0.0, 122.10000000000001)

In [13]: for bar, frequency in zip(axes.patches, frequencies):
   ....:     text_x = bar.get_x() + bar.get_width() / 2.0
   ....:     text_y = bar.get_height()
   ....:     text = f'{frequency:,}
   ....:             {frequency / len(rolls):.3%}'
   ....:     axes.text(text_x, text_y, text,
   ....:                  fontsize=11, ha='center', va='bottom')
```

To make room for the text above the bars, snippet [12] scales the y-axis by 10%. We chose this value via experimentation. The Axes object’s `set_xlim` method has many optional keyword arguments. Here, we use only `top` to change the maximum value represented by the y-axis. We multiplied the largest frequency by 1.10 to ensure that the y-axis is 10% taller than the tallest bar.

Finally, snippet [13] displays each bar’s frequency value and percentage of the total rolls. The axes object’s `patches` collection contains two-dimensional colored shapes that represent the plot’s bars. The for statement uses `zip` to iterate through the patches and their corresponding frequency values. Each iteration unpacks into `bar` and `frequency` one of the tuples `zip` returns. The for statement’s suite operates as follows:

- The first statement calculates the center x-coordinate where the text will appear. We calculate this as the sum of the bar’s left-edge x-coordinate (`bar.get_x()`) and half of the bar’s width (`bar.get_width() / 2.0`).
- The second statement gets the y-coordinate where the text will appear—`bar.get_y()` represents the bar’s top.
- The third statement creates a two-line string containing that bar’s frequency and the corresponding percentage of the total die rolls.
• The last statement calls the `Axes` object’s `text` method to display the text above the bar. This method’s first two arguments specify the text’s `x–y` position, and the third argument is the text to display. The keyword argument `ha` specifies the *horizontal alignment*—we centered text horizontally around the `x`-coordinate. The keyword argument `va` specifies the *vertical alignment*—we aligned the bottom of the text with at the `y`-coordinate. The final bar plot is shown below:

![Bar Plot](image)

**Rolling Again and Updating the Bar Plot—Introducing IPython Magics**

Now that you’ve created a nice bar plot, you probably want to try a different number of die rolls. First, clear the existing graph by calling Matplotlib’s `cla` (clear axes) function:

```python
In [14]: plt.cla()
```

IPython provides special commands called *magics* for conveniently performing various tasks. Let’s use the `%recall magic` to get snippet [5], which created the `rolls` list, and place the code at the next `In []` prompt:

```python
In [15]: %recall 5
```

```python
In [16]: rolls = [random.randrange(1, 7) for i in range(600)]
```

You can now edit the snippet to change the number of rolls to 60000, then press *Enter* to create a new list:

```python
In [16]: rolls = [random.randrange(1, 7) for i in range(60000)]
```

Next, recall snippets [6] through [13]. This displays all the snippets in the specified range in the next `In []` prompt. Press *Enter* to re-execute these snippets:

```python
In [17]: %recall 6-13
```

```python
In [18]: values, frequencies = np.unique(rolls, return_counts=True) 
   ...: title = f'Rolling a Six-Sided Die {len(rolls):,} Times' 
   ...: sns.set_style('whitegrid')
   ...: axes = sns.barplot(x=values, y=frequencies, palette='bright')
   ...: axes.set_title(title)
   ...: axes.set(xlabel='Die Value', ylabel='Frequency')
   ...: axes.set_ylim(top=max(frequencies) * 1.10)
```
The updated bar plot is shown below:

```
for bar, frequency in zip(axes.patches, frequencies):
    text_x = bar.get_x() + bar.get_width() / 2.0
    text_y = bar.get_height()
    text = f'{frequency:,}
         {frequency / len(rolls):.3%}'
    axes.text(text_x, text_y, text,
              fontsize=11, ha='center', va='bottom')
```

### Saving Snippets to a File with the %save Magic

Once you’ve interactively created a plot, you may want to save the code to a file so you can turn it into a script and run it in the future. Let’s use the %save magic to save snippets 1 through 13 to a file named RollDie.py. IPython indicates the file to which the lines were written, then displays the lines that it saved:

```
In [19]: %save RollDie.py 1-13
The following commands were written to file 'RollDie.py':
import matplotlib.pyplot as plt
import numpy as np
import random
import seaborn as sns
rolls = [random.randrange(1, 7) for i in range(600)]
values, frequencies = np.unique(rolls, return_counts=True)
title = f'Rolling a Six-Sided Die {len(rolls):,} Times'
sns.set_style("whitegrid")
axes = sns.barplot(values, frequencies, palette='bright')
axes.set_title(title)
axes.set(xlabel='Die Value', ylabel='Frequency')
axes.set_ylim(top=max(frequencies) * 1.10)
for bar, frequency in zip(axes.patches, frequencies):
    text_x = bar.get_x() + bar.get_width() / 2.0
    text_y = bar.get_height()
    text = f'{frequency:,}
         {frequency / len(rolls):.3%}'
    axes.text(text_x, text_y, text,
               fontsize=11, ha='center', va='bottom')
```
Command-Line Arguments; Displaying a Plot from a Script

Provided with this chapter’s examples is an edited version of the RollDie.py file you saved above. We added comments and a two modifications so you can run the script with an argument that specifies the number of die rolls, as in:

```
ipython RollDie.py 600
```

The Python Standard Library’s `sys` module enables a script to receive command-line arguments that are passed into the program. These include the script’s name and any values that appear to the right of it when you execute the script. The sys module’s `argv` list contains the arguments. In the command above, `argv[0]` is the string ‘RollDie.py’ and `argv[1]` is the string ‘600’. To control the number of die rolls with the command-line argument’s value, we modified the statement that creates the `rolls` list as follows:

```
rolls = [random.randrange(1, 7) for i in range(int(sys.argv[1]))]
```

Note that we converted the `argv[1]` string to an int.

Matplotlib and Seaborn do not automatically display the plot for you when you create it in a script. So at the end of the script we added the following call to Matplotlib’s `show` function, which displays the window containing the graph:

```
plt.show()
```

5.18 Wrap-Up

This chapter presented more details of the list and tuple sequences. You created lists, accessed their elements and determined their length. You saw that lists are mutable, so you can modify their contents, including growing and shrinking the lists as your programs execute. You saw that accessing a nonexistent element causes an `IndexError`. You used for statements to iterate through list elements.

We discussed tuples, which like lists are sequences, but are immutable. You unpacked a tuple’s elements into separate variables. You used `enumerate` to create an iterable of tuples, each with a list index and corresponding element value.

You learned that all sequences support slicing, which creates new sequences with subsets of the original elements. You used the `del` statement to remove elements from lists and delete variables from interactive sessions. We passed lists, list elements and slices of lists to functions. You saw how to search and sort lists, and how to search tuples. We used list methods to insert, append and remove elements, and to reverse a list’s elements and copy lists.

We showed how to simulate stacks with lists. We used the concise list-comprehension notation to create new lists. We used additional built-in methods to sum list elements, iterate backward through a list, find the minimum and maximum values, filter values and map values to new values. We showed how nested lists can represent two-dimensional tables in which data is arranged in rows and columns. You saw how nested for loops process two-dimensional lists.

The chapter concluded with an Intro to Data Science section that presented a die-rolling simulation and static visualizations. A detailed code example used the Seaborn and Matplotlib visualization libraries to create a static bar plot visualization of the simulation’s final results. In the next Intro to Data Science section, we use a die-rolling simulation with a dynamic bar plot visualization to make the plot “come alive.”
In the next chapter, “Dictionaries and Sets,” we’ll continue our discussion of Python’s built-in collections. We’ll use dictionaries to store unordered collections of key–value pairs that map immutable keys to values, just as a conventional dictionary maps words to definitions. We’ll use sets to store unordered collections of unique elements.

In the “Array-Oriented Programming with NumPy” chapter, we’ll discuss NumPy’s ndarray collection in more detail. You’ll see that while lists are fine for small amounts of data, they are not efficient for the large amounts of data you’ll encounter in big data analytics applications. For such cases, the NumPy library’s highly optimized ndarray collection should be used. ndarray (n-dimensional array) can be much faster than lists. We’ll run Python profiling tests to see just how much faster. As you’ll see, NumPy also includes many capabilities for conveniently and efficiently manipulating arrays of many dimensions. In big data analytics applications, the processing demands can be humongous, so everything we can do to improve performance significantly matters. In our “Big Data: Hadoop, Spark, NoSQL and IoT” chapter, you’ll use one of the most popular high-performance big-data databases—MongoDB.2

2. The database’s name is rooted in the word “humongous.”
Symbols
\^ regex metacharacter 206, 208
\^ set difference operator 150
\^= set symmetric difference 151
\_ (digit separator) 77
\_ SQL wildcard character 512
\_, (comma) in singleton tuple 107
\_: (colon) 44
\! = inequality operator 41, 45
\? to access help in IPython 74
\?? to access help in IPython (include source code) 74
\_ regular expression metacharacter 210
\\ single-quote-character escape sequence 37
\'relu\' (Rectified Linear Unit) activation function 475
\" double-quote-character escape sequence 37
\"is-a\" relationships 267
\{ ( and ) regex metacharacters 209
\[ ] regex character class 205
\{ subscription operator 103, 105
\{ for creating a dictionary 138
\{ placeholder in a format string 196
\{ (n,) quantifier (regex) 206
\{ (n,m) quantifier (regex) 207
\@-mentions 337, 353
\* multiplication operator 33, 45
\* operator for unpacking an iterable into function arguments 86
\* quantifier (regex) 206
\* SQL wildcard character 509
\* string repetition operator 110, 196
\** exponentiation operator 33, 45
\*= for lists 116
\+/ true division operator 33, 45
\// floor division operator 33, 45
\\ continuation character 38, 44
\\ escape character 37
\\ regex metacharacter 205
\\\ backslash character escape sequence 37
\\D regex character class 205
\\d regex character class 205
\\n newline escape sequence 37
\\S regex character class 205
\\s regex character class 205
\\t horizontal tab 37
\\t tab escape sequence 37
\\w regex character class 205
\\w regex character class 205
& bitwise AND operator 185
& set intersection operator 150
&= set intersection augmented assignment 151
# comment character 43
% remainder operator 33, 35, 45
% SQL wildcard character 512
+ addition operator 33, 45
+ subtraction operator 33, 45
+ operator for sequence concatenation 105
+ quantifier (regex) 206
- set difference operator 150
+ string concatenation operator 196
++ augmented assignment statement 57, 104
< less-than operator 41, 45
<= less-than-or-equal-to operator 41, 45
= assignment symbol 32, 44
= set difference augmented assignment 151
== equality operator 41, 44, 45
> greater-than operator 41, 45
>= greater-than-or-equal-to operator 41, 45
| (bitwise OR operator) 185
| set union operator 150
|= set union augmented assignment 151
$ regex metacharacter 209
Numerics
0D tensor 465
1D tensor 465
2D tensor 466
3D tensor 466
4D tensor 466
5D tensor 466
A
'a' file-open mode 226
'\a' file-open mode 226
abbreviating an assignment expression 57
abs built-in function 83
absence of a value (None) 74
absolute value 83
accept method of a socket 554
access token (Twitter) 336, 341
access token secret (Twitter) 336, 341
Account class 246, 287
Accounts and Users API (Twitter) 334
accounts-receivable file 219
accuracy 496
accuracy of a model 480
ACID (Atomicity, Consistency, Isolation, Durability) 519
acquire resources 220
activate a neuron 464
activation function 465, 468
\texttt{relu} (Rectified Linear Unit) 475
sigmoid 496
softmax 478
adam optimizer 480
add
method of set 152
universal function (NumPy) 170, 171
__add__ special method of class
object 276, 278
add_to method of class Marker 366
addition 33, 36
augmented assignment (++=) 57
adjective 308
algebraic expression 35
alignment 194
a1 built-in function 117
alphabetic characters 202
AlphaGo 27
alphanumeric character 202, 205
AlphaZero 27
Amazon DynamoDB 517
Amazon EMR 533
Ambani 532
Anaconda Python distribution 6, 9
 base environment command 462
conda activate command 463
conda create command 463
conda deactivate command 463
environment 462
install xxxiv
installer xxxiv
NumPy preinstalled 160
packages installed xxxiv
 update xxxiv
Anaconda Command Prompt, Windows xxiv
analyze text for tone 378
anchor (regex) 208, 209
and Boolean operator 65, 66
truth table 65
animated visualization 153
animation frame 153, 154
animation module (Matplotlib) 153, 157
 FuncAnimation function 153, 156, 157, 158
anomaly detection 24
anonymous function 123
Anscombe’s quartet xx
answering natural language questions 329
antonyms 305, 315, 316
any built-in function 117
Apache Hadoop xix, 16, 503, 530
Apache HBase 531
Apache Ignite (NewSQL) 520
Apache Kafka 562
Apache Mesos 541
Apache OpenNLP 328
Apache Spark 16, 503
API class (Tweepy) 341, 342
 followers method 344
 followers_ids method 345
friends method 346
 get_user method 342
 home_timeline method 347
lookup_users method 346
 me method 344
 search method 347
 API class (Tweepy) (cont.)
trends_available method 350
trends_closest method 351
trends_place method 351
user_timeline method 346
API key (Twitter) 336, 341
API reference (IBM Watson) 394
API secret key (Twitter) 336, 341
app rate limit (Twitter API) 334
append method of list 117, 119
approximating a floating-point number 61
arange function (NumPy) 164
arccos universal function (NumPy) 171
arcsin universal function (NumPy) 171
arctan universal function (NumPy) 171
*args parameter for arbitrary argument lists 86
argv list of command-line arguments 135
argv[0] first command-line argument 135
arithmetic expressions 9
arithmetic on ndarray 167
arithmetic operator 33
 Decimal 62
 "arity" of an operator 277
ARPANET 560
array, JSON 224
array attributes (NumPy) 161
array function (NumPy) 161, 162
artificial general intelligence 26
artificial intelligence (AI) xxi, 26, 27
artificial neural network 463
artificial neuron in an artificial neural network 464
as clause of a with statement 220
as-a-service
big data (BDaaS) 504
Hadoop (Haas) 504
Hardware (Haas) 504
Infrastructure (laaS) 504
platform (PaaS) 504
software (SaaS) 504
Spark (SaaS) 504
storage (SaaS) 504
ascending order
ASC in SQL 512
sort 115, 146
assignment symbol ( = ) 32, 44
assisting people with disabilities 24
asterisk (*) multiplication operator 33, 45
asterisk (*) SQL wildcard character 509
astype method of class Series 522
asynchronous 379
asynchronous tweet stream 358
at attribute of a DataFrame 185
atomicity 519
attribute 4
 internal use only 250
 of a class 3, 247
 of an array 161
 of an object 4
 publicly accessible 250
AudioSegment class
 from pydub module 393
 fromwav method 393
 augmented assignment addition (+) 57, 104
Authentication API (Twitter) 334
author_ISBN table of books database 508, 509
authors table of books database 508
auto insurance risk prediction 24
autocompleted value 508, 515
Auto-Keras automated deep learning library 460, 498
automated
closed captioning 24, 490
image captions 24
investing 24
machine learning (AutoML) 498
AutoML 460
autonomous ships 24
average time 166
Averaged Perceptron Tagger 306
Axes class (Matplotlib) 131
 ishow method 264
 set method 131
 set_xlim method 132
text method 131, 133
Axes3D class (Matplotlib) 442
axis=1 keyword argument of Dataframe method sort_index 187
Azure HDInsight (Microsoft) 503

B
 b prefix for a byte string 393
backpropagation 465
backslash (\) escape character 37
bad data values 211
balanced classes in a dataset 411
bar chart 319, 321
static 102
bar method of a DataFrame’s plot property 321
bar plot 128, 152, 153
barplot function (Seaborn) 131
BASE (Basic Availability, Soft-state, Eventual consistency) 520
base-10 number system 83
base case 93
base class 245
direct 267
indirect 267
base e 83
base environment in Anaconda 462
BaseBlob class from the textblob module 307
BaseException class 279
batch
interval in Spark streaming 558
of data in Hadoop 532
of streaming data in Spark 558
batch_size argument to a Keras model’s fit method 480
BDaaS (Big data as a Service) 504
behavior of a class 3
big data 22, 160
analytics 23
analytics in the cloud xxi
bimodal set of values 68
binary classification 490, 496
machine learning 403
binary file 219
binary number system 193
binary_crossentropy loss function 480, 496
bind a name to an object 45
bind method of a socket 554
Bing sentiment analysis 328
BitBucket 245
Bitcoin 21
biwise
AND operator (&) 185
OR operator (|) 185
bitwise_and universal function (NumPy) 171
bitwise_or universal function (NumPy) 171
bitwise_xor universal function (NumPy) 171
block
in a function 73, 74
vs. suite 73, 88
blockchain 21
books database 507
book-title capitalization 197
bool NumPy type 162
Boolean indexing (pandas) 185
Boolean operators 65
and 65
not 65, 66, 67
or 65, 66
Boolean values in JSON 224
brain mapping 24
break statement 64
broadcasting (NumPy) 168, 171
Brown Corpus (from Brown University) 306
brute force computing 26
building-block approach 4
built-in functions
abs 83
all 117
any 117
ever 109, 110
filter 122
float 41
frozenset 148
id 91
input 39
int 40, 41
len 68, 86, 103
list 109
map 123
max 48, 76, 86, 124
min 48, 76, 86, 124
open 220
ord 124
print 36
range 57, 60
repr 254
reversed 125
set 148
sorted 68, 115, 143
str 255
sum 68, 80, 86
super 272
tuple 109
zip 125
built-in namespace 291
built-in types
dict (dictionary) 138
float 45, 62
int 45, 62
set 138, 147
str 45, 62
Bunch class from sklearn.utils 426
data attribute 407, 428
Bunch class from sklearn.utils (cont.)
DESCR attribute 406, 427
feature_names attribute 428
target attribute 407, 428
byte string 393
C
c presentation type 193
C programming language 162
cadence, voice 377
calendar module 82
California Housing dataset 426
call-by-reference 90
call-by-value 90
callback (Keras) 488
caller 73
caller identification 24
CamelCase naming convention 84
cancer diagnosis 24
capitalization
book title 197
sentence 197
capitalize method of a string 197
carbon emissions reduction 24
card class 258, 259, 282
card images 258
caret (^) regex metacharacter 206
case insensitive 208
case-insensitive sort 345
case sensitive 33, 208
catching multiple exceptions in one except clause 230
categorical data 472, 494
categorical features in machine learning datasets 408
categorical_crossentropy loss function 480
%cd magic 167
cell (ceiling) function 83
cell universal function (NumPy) 171
cell in a Jupyter Notebook 14
central nervous system 463
centroid 442, 450
chained method calls 164
circuit in Pub/Sub system 562
class character (regular expressions) 205
custom 205
cart xix
chatbots 376
checkpoint method of a StreamingContext 558
checkpointing in Spark 558
<table>
<thead>
<tr>
<th>Page</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>576</td>
<td></td>
</tr>
<tr>
<td></td>
<td>chess 26</td>
</tr>
<tr>
<td></td>
<td>Chinese (simplified) 312</td>
</tr>
<tr>
<td></td>
<td>choice function from the numpy.random module 469</td>
</tr>
<tr>
<td></td>
<td>choropleth 527</td>
</tr>
<tr>
<td></td>
<td>chunking text 306</td>
</tr>
<tr>
<td></td>
<td>CIFAR10 dataset (Keras) 462</td>
</tr>
<tr>
<td></td>
<td>CIFAR100 dataset (Keras) 462</td>
</tr>
<tr>
<td></td>
<td>cla function of matplotlib.pyplot module 133, 156</td>
</tr>
<tr>
<td></td>
<td>class 3, 81</td>
</tr>
<tr>
<td></td>
<td>attribute 247</td>
</tr>
<tr>
<td></td>
<td>class keyword 248</td>
</tr>
<tr>
<td></td>
<td>client code 250</td>
</tr>
<tr>
<td></td>
<td>data attribute 248</td>
</tr>
<tr>
<td></td>
<td>definition 248</td>
</tr>
<tr>
<td></td>
<td>header 248</td>
</tr>
<tr>
<td></td>
<td>instance variable 4</td>
</tr>
<tr>
<td></td>
<td>library 245</td>
</tr>
<tr>
<td></td>
<td>method 281</td>
</tr>
<tr>
<td></td>
<td>namespace 292</td>
</tr>
<tr>
<td></td>
<td>object 249, 269</td>
</tr>
<tr>
<td></td>
<td>property 251, 253</td>
</tr>
<tr>
<td></td>
<td>@property decorator 253</td>
</tr>
<tr>
<td></td>
<td>@propertyname.setter decorator 253</td>
</tr>
<tr>
<td></td>
<td>public interface 255</td>
</tr>
<tr>
<td></td>
<td>variable 259, 282</td>
</tr>
<tr>
<td></td>
<td>class attribute 259</td>
</tr>
<tr>
<td></td>
<td>in a data class 283</td>
</tr>
<tr>
<td></td>
<td>class average for arbitrary number of grades 59</td>
</tr>
<tr>
<td></td>
<td>class average problem 58, 59</td>
</tr>
<tr>
<td></td>
<td>class libraries xxii</td>
</tr>
<tr>
<td></td>
<td>classification (machine learning) 400, 401, 403</td>
</tr>
<tr>
<td></td>
<td>algorithm 404</td>
</tr>
<tr>
<td></td>
<td>binary classification 403</td>
</tr>
<tr>
<td></td>
<td>handwritten digits 467</td>
</tr>
<tr>
<td></td>
<td>metrics 415</td>
</tr>
<tr>
<td></td>
<td>multi-classification 403</td>
</tr>
<tr>
<td></td>
<td>probabilities (deep learning) 478</td>
</tr>
<tr>
<td></td>
<td>classification report (scikit-learn) 416</td>
</tr>
<tr>
<td></td>
<td>precision 416</td>
</tr>
<tr>
<td></td>
<td>recall 416</td>
</tr>
<tr>
<td></td>
<td>support 416</td>
</tr>
<tr>
<td></td>
<td>classification_report function from the sklearn.metrics module 415</td>
</tr>
<tr>
<td></td>
<td>classifier 378</td>
</tr>
<tr>
<td></td>
<td>classify handwriting 24</td>
</tr>
<tr>
<td></td>
<td>ClassVar type annotation from the typing module 282, 283</td>
</tr>
<tr>
<td></td>
<td>cleaning data 204, 239, 366</td>
</tr>
<tr>
<td></td>
<td>clear axes 156</td>
</tr>
<tr>
<td></td>
<td>clear method of dictionary 139</td>
</tr>
<tr>
<td></td>
<td>of list 118</td>
</tr>
<tr>
<td></td>
<td>of set 152</td>
</tr>
<tr>
<td></td>
<td>client of a class 250, 257</td>
</tr>
<tr>
<td></td>
<td>client/server app 251</td>
</tr>
<tr>
<td></td>
<td>server 551</td>
</tr>
<tr>
<td></td>
<td>client/server networking 551</td>
</tr>
<tr>
<td></td>
<td>close method of a file object 220</td>
</tr>
<tr>
<td></td>
<td>of a socket 554</td>
</tr>
<tr>
<td></td>
<td>of a sqlite3 Connection 516</td>
</tr>
<tr>
<td></td>
<td>of an object that uses a system resource 220</td>
</tr>
<tr>
<td></td>
<td>of class Stream 393</td>
</tr>
<tr>
<td></td>
<td>closed captioning 329, 490</td>
</tr>
<tr>
<td></td>
<td>closures 95</td>
</tr>
<tr>
<td></td>
<td>cloud xix, 16, 334, 374</td>
</tr>
<tr>
<td></td>
<td>IBM Cloud account 374</td>
</tr>
<tr>
<td></td>
<td>cloud-based services 16, 223</td>
</tr>
<tr>
<td></td>
<td>Cloudera CDH 533</td>
</tr>
<tr>
<td></td>
<td>cluster 531</td>
</tr>
<tr>
<td></td>
<td>node 531</td>
</tr>
<tr>
<td></td>
<td>clusters of computers 25</td>
</tr>
<tr>
<td></td>
<td>CNN (convolutional neural network) 467</td>
</tr>
<tr>
<td></td>
<td>CNTK (Microsoft Cognitive Toolkit) 8, 458, 462</td>
</tr>
<tr>
<td></td>
<td>code 5</td>
</tr>
<tr>
<td></td>
<td>coeff_attribute of a LinearRegression estimator 423</td>
</tr>
<tr>
<td></td>
<td>coefficient of determination (R2 score) 437</td>
</tr>
<tr>
<td></td>
<td>cognitive computing xvi, 374, 378</td>
</tr>
<tr>
<td></td>
<td>Cognos Analytics (IBM) 381</td>
</tr>
<tr>
<td></td>
<td>collaborative filtering 329</td>
</tr>
<tr>
<td></td>
<td>collection non-sequence 138</td>
</tr>
<tr>
<td></td>
<td>sequence 138</td>
</tr>
<tr>
<td></td>
<td>unordered 139</td>
</tr>
<tr>
<td></td>
<td>Collection class of the pymongo module 524</td>
</tr>
<tr>
<td></td>
<td>count_documents method 525</td>
</tr>
<tr>
<td></td>
<td>insert_one method 524</td>
</tr>
<tr>
<td></td>
<td>collections 102</td>
</tr>
<tr>
<td></td>
<td>collections module 7, 82, 145, 280</td>
</tr>
<tr>
<td></td>
<td>namedtuple function 280</td>
</tr>
<tr>
<td></td>
<td>color map 410</td>
</tr>
<tr>
<td></td>
<td>Matplotlib 410</td>
</tr>
<tr>
<td></td>
<td>column in a database table 507, 508</td>
</tr>
<tr>
<td></td>
<td>in a multi-dimensional list 126</td>
</tr>
<tr>
<td></td>
<td>columnar database (NoSQL) 517, 518</td>
</tr>
<tr>
<td></td>
<td>column-oriented database 517, 518</td>
</tr>
<tr>
<td></td>
<td>comma (,) format specifier 130</td>
</tr>
<tr>
<td></td>
<td>comma-separated list of arguments 73</td>
</tr>
<tr>
<td></td>
<td>comma-separated-value (CSV) files 82</td>
</tr>
<tr>
<td></td>
<td>command-line arguments 135</td>
</tr>
<tr>
<td></td>
<td>argv 135</td>
</tr>
<tr>
<td></td>
<td>argv[0] 135</td>
</tr>
<tr>
<td></td>
<td>comma-separated-value (CSV) files 7</td>
</tr>
<tr>
<td></td>
<td>code 43</td>
</tr>
<tr>
<td></td>
<td>comment character (#) 43</td>
</tr>
<tr>
<td></td>
<td>CommissionEmployee class 268</td>
</tr>
<tr>
<td></td>
<td>common programming errors xxviii</td>
</tr>
<tr>
<td></td>
<td>comparison operators 41</td>
</tr>
<tr>
<td></td>
<td>compile method of class Sequential 480</td>
</tr>
<tr>
<td></td>
<td>Complex class 277</td>
</tr>
<tr>
<td></td>
<td>complex condition 65</td>
</tr>
<tr>
<td></td>
<td>component 3, 245</td>
</tr>
<tr>
<td></td>
<td>composite primary key 509, 510</td>
</tr>
<tr>
<td></td>
<td>composition (has a “relationship”) 249, 274</td>
</tr>
<tr>
<td></td>
<td>compound interest 63</td>
</tr>
<tr>
<td></td>
<td>computer vision 24</td>
</tr>
<tr>
<td></td>
<td>computer-vision applications 26</td>
</tr>
<tr>
<td></td>
<td>concatenate sequences 105</td>
</tr>
<tr>
<td></td>
<td>concatenate strings separated by whitespace 144</td>
</tr>
<tr>
<td></td>
<td>concurrent execution 546</td>
</tr>
<tr>
<td></td>
<td>concurrent programming 7</td>
</tr>
<tr>
<td></td>
<td>conda activate command 463</td>
</tr>
<tr>
<td></td>
<td>conda command xxxiv</td>
</tr>
<tr>
<td></td>
<td>conda create command 463</td>
</tr>
<tr>
<td></td>
<td>conda deactivate command 463</td>
</tr>
<tr>
<td></td>
<td>conda package manager xxxiv</td>
</tr>
<tr>
<td></td>
<td>condition 41</td>
</tr>
<tr>
<td></td>
<td>None evaluates to False 74</td>
</tr>
<tr>
<td></td>
<td>conditional expression 53</td>
</tr>
<tr>
<td></td>
<td>operators 65</td>
</tr>
<tr>
<td></td>
<td>confidence interval 299</td>
</tr>
<tr>
<td></td>
<td>confusion matrix 414</td>
</tr>
<tr>
<td></td>
<td>as a heat map 416</td>
</tr>
<tr>
<td></td>
<td>confusion_matrix function of the sklearn.metrics module 414</td>
</tr>
<tr>
<td></td>
<td>conjunction, subordinating 308</td>
</tr>
<tr>
<td></td>
<td>conll2000 (Conference on Computational Natural Language Learning 2000) 306</td>
</tr>
<tr>
<td></td>
<td>connect function from the sqlite3 module 508</td>
</tr>
</tbody>
</table>
Index

data science libraries
  Gensim 9
  Matplotlib 8
  NLTK 9
  NumPy 8
  pandas 8
  scikit-learn 8
  SciPy 8
  Seaborn 8
  StatsModels 8
  TensorFlow 8
  TextBlob 9
  Theano 8
data sources 329
data visualization 24
data warehouse 532
data wrangling 192, 210
database 502, 506, 511
Database Application Programming Interface (DB-API) 507
Database class of the pymongo module 523
database management system (DBMS) 506
Databricks 542
@dataclass decorator from the module dataclasses 282
dataclasses module 281, 282
@dataclass decorator 282
DataFrame (pandas) 178, 182, 192, 211, 213, 214, 366
  at attribute 185
  describe method 186
  dropna method 366
  groupby method 527, 529
  head method 238
  hist method 240
  iat attribute 185
  iloc attribute 183
  index attribute 182
  index keyword argument 182
  iteritems method 366
  loc attribute 183
  plot method 294
  plot property 321
  sample method 430
  sort_index method 187
  sort_values method 188
  sum method 527
  T attribute 187
tail method 238
to_csv method 238
  transpose rows and columns 187
DataFrame (Spark) 555, 557
  createOrReplaceTempView method 557
  pyspark.sql module 555, 557
data-interchange format, JSON 223
dataset
  California Housing 426
  CIFAR10 462
  CIFAR100 462
  Digits 403
  EMNIST 485
  Fashion-MNIST 462
  ImageNet 477, 478
  IMDb Movie reviews 462
  Iris 442
  MNIST digits 461, 467
  natural language 329
  Titanic disaster 237, 238
  UCI ML hand-written digits 406
  date and time manipulations 7, 82
datetime module 7, 82, 256
  DB-API (Database Application Programming Interface) 507
  Db2 (IBM) 506
  DBMS (database management system) 506
debug 48
diagnosing breast cancer 24
diagnostic medicine 24
dice game 78
deserailizing data 224
descriptive statistics 46, 67, 97, 179, 186, 239
describing a User (Twitter) 343
determiner 308
diagnose medical conditions 26
diagnosing breast cancer 24
diagnosing heart disease 24
diagnostic medicine 24
dice game 78
dict method of class Text( 324
dictionary 95
diagram 138
diagram built-in type 138
  clear method 139
get method 141
  immutable keys 138
  items method 140
  keys method 141
exception classes
EZDL automated deep learning estimator (model) in scikit-learn
Ethereum 21
ETL (extract, transform, load) 532
eval built-in function 255
evaluate method of class
Sequential 482
evaluation order 10
evenly-spaced values 164
exabytes (EB) 19
exaflops 21
factorial 21
fabs
fabs
f
f
F
F
extended_tweet
extend
exp
exp
execute
exceptions
Exception
except
evaluate
eval
escape character 37, 515
escape sequence 37, 203
estimator (model) in scikit-learn 405, 422
Facebook 333
Facial Recognition 24
factorial 93, 94
False 41, 50, 51
fargs keyword argument of FuncAnimation 158
Fashion-MNIST dataset (Keras) 462
fatal
logic error 55, 59
runtime error 12
fault tolerance 218
Spark streaming 558
feature in a dataset 211
feature map 475
feature_names attribute of a Bunch 428
feed-forward network 473
fetch_california_housing function from sklearn.datasets 426
field alignment 194
field width 63, 194
FIFO (first-in, first-out) order) 120
Figure class (Matplotlib) 131
tight_layout method 264
Figure function of matplotlib.pyplot.module 157
file 218
contents deleted 226
file object 219, 220
close method 220
in a for statement 221
read method 227
readline method 227
readlines method 221
seek method 221
standard 219
write method 220
writelines method 227
file-open mode 220
'a' (append) 226
'a+' (read and append) 226
'r' (read) 221, 226
'r+' (read and write) 226
'w' (write) 220, 226
'w+' (read and write) 226
file-position pointer 221
file/directory access 7
FileNotFoundException 227, 232
fill with 0s 195
filter sequence 121, 122
filter built-in function 95, 122
filter in convolution 474
filter method of class Stream 358
filter method of the RDD class 547
filter/map/reduce operations 178
finally clause 231
finally suite raising an exception 235
find string method 199
findall function of the module re 209
finditer function of the module re 209
fire hose (Twitter) 354
first 120
first-in, first-out (FIFO) order 120
fit method
batch_size argument 480
ePOCH argument 480
of a scikit-learn estimator 412, 434
of class Sequential 480
of the PCA estimator 452
of the TSNE estimator 440
validation_data argument 494
validation_split argument 481, 494
fit_transform method
of the PCA estimator 452
of the TSNE estimator 440
fitness tracking 24
flag value 59
flags keyword argument (regular expressions) 208
flat attribute of ndarray 163
flatMap method of DStream 559
flatMap method of the RDD class 546
Flatten class from the tensorflow.keras.layers module 477
flatten method of ndarray 176
Flesch Reading Ease readability formula 324, 325
Flesch-Kincaid readability formula 324, 325
float function 41
float type 33, 45, 62
float64 NumPy type 162, 163
floating-point number 10, 33, 34, 45
floor division 33, 34, 45
operator (//) 34
floor function of module math 83
floor universal function (NumPy) 171
FLOPS (floating-point operations per second) 20
flow of control 64
Flume 532
Flesch-Kincaid readability formula
Index

H

graph database 517, 519
edge 519
node 519
vertices 519
Graphics Processing Unit (GPU) 459, 466
greater universal function (NumPy) 171
greater_equal universal function (NumPy) 171
greater-than operator (>) 41
greedy evaluation 121
greedy quantifier
handwritten digits
Guido van Rossum 5, 7
"has a" relationship (composition)
Hadoop (Apache) xix, 124, 503, 530
as a Service (HaaS) 504
streaming 535
streaming mapper 535
streaming reducer 536
YARN ("yet another resource negotiator")
HBase 517, 531, 532
HDFS (Hadoop Distributed File System) 531, 558
HDInsight
(Microsoft Azure) 503
head method of a DataFrame 238
health outcome improvement 24
heat map 416
heatmap function (Seaborn visualization library) 416
help in IPython 74
heterogeneous data 106
hexadecimal number system 193
highest level of precedence 35
HIPAA (Health Insurance Portability and Accountability Act) 502
hist method of a DataFrame 240
histogram 109
histology magic 167
Hive 532
home timeline 347
home_timeline method of class API 347
homogeneous data 102, 103
homogeneous numeric data 177
horizontal stacking (ndarray) 177
horizontal tab (\t) 37
Hortonworks 533
hospital readmission reduction 24
hostname 554
hstack function (NumPy) 177
human genome sequencing 24
hyperparameter
in machine learning 405, 420
hyperparameter of a list comprehension
IBM Watson (cont.)
dashboard 375
deep learning 379
GitHub repository 394
Knowledge Studio 380
Language Translator service
377, 382, 383, 384, 385, 390
Lite tier 374
lite tiers xxvii
machine learning 380
Machine Learning service 380
Natural Language Classifier service 378
Natural Language Understanding service 377
Personality Insights service 378
Python SDK xxvi, 375
service documentation 394
Speech to Text service 377, 382, 383, 384, 385, 386, 387, 391
Text to Speech service 377, 382, 383, 384, 385
Tone Analyzer service 378
use cases 374
Visual Recognition service 376
Watson Assistant service 376
Watson Developer Cloud
Python SDK 381, 385, 394
Watson Discovery service 378
Watson Knowledge Catalog 380
YouTube channel 395
H5 file extension for Hierarchical Data Format files 485
Hadoop (Apache) xix, 124, 503, 530
as a Service (HaaS) 504
streaming 535
streaming mapper 535
streaming reducer 536
YARN ("yet another resource negotiator")
HBase 517, 531, 532
HDFS (Hadoop Distributed File System) 531, 558
HDInsight
(Microsoft Azure) 503
head method of a DataFrame 238
health outcome improvement 24
heat map 416
heatmap function (Seaborn visualization library) 416
help in IPython 74
heterogeneous data 106
hexadecimal number system 193
highest level of precedence 35
HIPAA (Health Insurance Portability and Accountability Act) 502
hist method of a DataFrame 240
histogram 109
histology magic 167
Hive 532
home timeline 347
home_timeline method of class API 347
homogeneous data 102, 103
homogeneous numeric data 177
horizontal stacking (ndarray) 177
horizontal tab (\t) 37
Hortonworks 533
hospital readmission reduction 24
hostname 554
hstack function (NumPy) 177
human genome sequencing 24
hyperparameter
in machine learning 405, 420
tuning 406, 420, 497
tuning (automated) 406
hypot universal function (NumPy) 171
I

__iadd__ special method of class object 279
iat attribute of a DataFrame 185
IBM Cloud account 374, 375, 382
IBM Cloud console 375
IBM Cloud dashboard 376
IBM Cognos Analytics 381
IBM Db2 506
IBM DeepBlue 26
IBM Watson xxvi, 16, 27, 334, 374
Analytics Engine 533
API reference 394
IBM Watson (cont.)
dashboard 375
deep learning 379
GitHub repository 394
Knowledge Studio 380
Language Translator service
377, 382, 383, 384, 385, 390
Lite tier 374
lite tiers xxvii
machine learning 380
Machine Learning service 380
Natural Language Classifier service 378
Natural Language Understanding service 377
Personality Insights service 378
Python SDK xxvi, 375
service documentation 394
Speech to Text service 377, 382, 383, 384, 385, 386, 387, 391
Text to Speech service 377, 382, 383, 384, 385
Tone Analyzer service 378
use cases 374
Visual Recognition service 376
Watson Assistant service 376
Watson Developer Cloud
Python SDK 381, 385, 394
Watson Discovery service 378
Watson Knowledge Catalog 380
YouTube channel 395
i0 built-in function 91, 173
i0 property of a user (Twitter) 342
IDE (integrated development environment) 11
identifiers 33
identity of an object 91
identity theft prevention 24
if clause of a list comprehension
121
if statement 41, 44, 50, 51
if...elif...else statement 50, 54
if...else statement 50, 52
IGNORECASE regular expression flag 208
i0c attribute of a DataFrame 183
image 376
image (Docker) 542
Image class of the IPython.display module 479
imageio module
i0read function 322
ImageNet dataset 477, 478
...


**I**

I/O (input/output) 411
iterable 56, 76, 105, 121
iterating over lines in a file 221
is-a relationship 267

**J**

JavaScript Object Notation (JSON) 7, 337
Jeopardy! dataset 329
join string method 200
joining 510
joining database tables 510, 514
joinpath method of class Path 263, 264

**K**

k-fold cross-validation 417, 498
Kafka 532
Keras 380
Keras (cont.)
metrics 480
MNIST digits dataset 461
optimizers 480
reproducibility 467
summary method of a model 478
TensorBoard callback 488
Keras deep learning library 458
kernel
in a convolutional layer 473
key 116
key-value
database 517
pair 138
KeyboardInterrupt exception 279
keys
API keys 335
credentials 335
keys method of dictionary 141
keyword 41, 44, 51
and 65, 66
argument 56, 85
break 64
class 248
continue 64
def 73
e1f 51
e1lse 51
False 50, 51
for 50, 51, 55, 55, 57
from 62
if 50, 51
if...e1l1f...e1lse 50, 54
if...e1lse 50, 52
import 62, 62
in 50, 51, 55, 57
lambda 123
not 65, 66, 67
or 65, 66
True 50, 51
while 50, 51

**K**

kFold class
sklearn.model_selection 417, 419, 438
Kitematic (Docker GUI app) 545
k-means clustering algorithm 442, 450

**K**

Keras deep learning library 458
kernel
in a convolutional layer 473
key 116
key-value
database 517
pair 138
KeyboardInterrupt exception 279
keys
API keys 335
credentials 335
keys method of dictionary 141
keyword 41, 44, 51
and 65, 66
argument 56, 85
break 64
class 248
continue 64
def 73
e1f 51
e1lse 51
False 50, 51
for 50, 51, 55, 55, 57
from 62
if 50, 51
if...e1l1f...e1lse 50, 54
if...e1lse 50, 52
import 62, 62
in 50, 51, 55, 57
lambda 123
not 65, 66, 67
or 65, 66
True 50, 51
while 50, 51

**K**

kFold class
sklearn.model_selection 417, 419, 438
Kitematic (Docker GUI app) 545
k-means clustering algorithm 442, 450

**K**

Keras deep learning library 458
kernel
in a convolutional layer 473
key 116
key-value
database 517
pair 138
KeyboardInterrupt exception 279
keys
API keys 335
credentials 335
keys method of dictionary 141
keyword 41, 44, 51
and 65, 66
argument 56, 85
break 64
class 248
continue 64
def 73
e1f 51
e1lse 51
False 50, 51
for 50, 51, 55, 55, 57
from 62
if 50, 51
if...e1l1f...e1lse 50, 54
if...e1lse 50, 52
import 62, 62
in 50, 51, 55, 57
lambda 123
not 65, 66, 67
or 65, 66
True 50, 51
while 50, 51

**K**

Keras deep learning library 458
kernel
in a convolutional layer 473
key 116
key-value
database 517
pair 138
KeyboardInterrupt exception 279
keys
API keys 335
credentials 335
keys method of dictionary 141
keyword 41, 44, 51
and 65, 66
argument 56, 85
break 64
class 248
continue 64
def 73
e1f 51
e1lse 51
False 50, 51
for 50, 51, 55, 55, 57
from 62
if 50, 51
if...e1l1f...e1lse 50, 54
if...e1lse 50, 52
import 62, 62
in 50, 51, 55, 57
lambda 123
not 65, 66, 67
or 65, 66
True 50, 51
while 50, 51

**K**

Keras deep learning library 458
kernel
in a convolutional layer 473
key 116
key-value
database 517
pair 138
KeyboardInterrupt exception 279
keys
API keys 335
credentials 335
keys method of dictionary 141
keyword 41, 44, 51
and 65, 66
argument 56, 85
break 64
class 248
continue 64
def 73
e1f 51
e1lse 51
False 50, 51
for 50, 51, 55, 55, 57
from 62
if 50, 51
if...e1l1f...e1lse 50, 54
if...e1lse 50, 52
import 62, 62
in 50, 51, 55, 57
lambda 123
not 65, 66, 67
or 65, 66
True 50, 51
while 50, 51
L
L1 regularization 476
L2 regularization 476
label_ property of a spaCy Span 327
labeled data 400, 403
lambda expression 95, 123, 345
lambda keyword 123
language codes 312
language detection 305
language translation 24, 305
LanguageTranslator service (IBM Watson) 377, 382, 383, 384, 385, 390
LanguageTranslatorV3 class from the
watson_developer_cloud module 385, 390
translate method 390
largest integer not greater than 83
Lasso estimator from
sklearn.linear_model 438
last-in, first-out (LIFO) order 119
latitude 363
Law of Large Numbers 2
law of large numbers xx, 102, 153, 154
layers 468
layers in a neural network 458, 464, 468
lazy estimator (scikit-learn) 412
lazy evaluation 95, 121, 123
Leaflet.js JavaScript mapping library 363, 364
leave interactive mode 10
left align (\<) in string formatting 64, 194
left-to-right evaluation 36, 45
left_shift universal function
(NumPy) 171
leftmost condition 66
legacy code 226
LEGGB (local, enclosing, global,
built-in) rule 292
lemmas method of class Synset 316
lemmatization 305, 309, 314
lemmatize method of class
Sentence 314
lemmatize method of class Word 314
len built-in function 68, 86, 103
length of a dictionary 139
less universal function (NumPy)
171
less_equal universal function
(NumPy) 171
less-than operator (\<) 41
less-than-or-equal-to operator (\<=) 41
lexicographical comparison 198
lexicographical order 125
libraries xix, 7
LIFO (last-in, first-out) order 119
LIKE operator (SQL) 512
linear regression xix, 293
multiple 421, 434
simple 420, 421
linear relationship 293, 294
LinearRegression estimator from
sklearn.linear_model 421, 422, 434
coeff_attribute 423
intercept_attribute 423
linguistic analytics 378
Linkedin 333
linregress function of SciPy’s
stats module 295, 298
linspace function (NumPy) 164
Linux Terminal or shell xxiv
lip reader technology 329
list 102
*= 116
append method 117, 119
clear method 118
copy method 119
extend method 118
index method 116
insert method 117
pop method 119
remove method 118
reverse method 119
list built-in function 109
list comprehension 95, 120, 130, 493
filter and map 124
for clause 121
if clause 121
list indexing in pandas 184
list method
sort 115
list of base-class objects 274
list sequence 56, 58
List type annotation from the
typing module 282, 283
listen method of a socket 554
listener for tweets from a stream 355
Lite tier (IBM Watson) 374
literal character 204
literal digits 204
load function from the json
module 224
load function of the spacy module
326
load magic 167
load_data function of the
tensorflow.keras.datasets.mnist
module 468, 491
load_digits function from
sklearn.datasets 406
load_iris function from
sklearn.datasets 444
load_model function of the
tensorflow.keras.models
module 485
loc attribute of a DataFrame 183
local
namespace 290
scope 87
variable 74, 75, 87
locale module 82
localization 82
location-based services 24
log (natural logarithm) function of
module math 83
log universal function (NumPy)
171
log10 (logarithm) function of
module math 83
logarithm 83
logic error 11, 55
fatal 59
logical_and function from the
math module 74
logical_or function from the
math module 75, 87
logical_not function from the
math module 75, 87
List type annotation from the
typing module 282, 283
look up users method of class API
346
loss 465
loss function 465, 480
binary_crossentropy 480, 496
categorical_crossentropy 480
deep learning 468
Keras 480
mean_squared_error 480
token method of a string 87, 125, 197

Index
ndarray (cont.)
  indexing 171
  itemsize attribute 162
  max method 169
  mean method 169
  min method 169
  ndim attribute 162
  ravel method 176
  reshape method 164, 175
  resize method 175
  shape attribute 162
  size attribute 162
  std method 169
  sum method 169
  var method 169
  view method 173

ndarray collection (NumPy)
  T attribute 177
  ndim attribute of ndarray 162
  negative sentiment 309

Neo4j 519
nest
  control statements 77
  for structure 127
  functions 292
  list 126
  loop 127
  parentheses 35

network (neural)
  layer 464, 468
  loss function 468
  model 468
  neuron 464
  optimizer 468
  weight 465
  activation 464
  in a neural network 464
  in biology 463

neutral sentiment 309

new pharmaceuticals 24

newline character (\n) 37

NewSQL database 503, 517, 520
  Apache Ignite 520
  Google Spanner 520
  MemSQL 520
  VoltDB 520

n-grams 305, 318
  ngrams method of class TextBlob 318

NLTK (Natural Language Toolkit)
  NLP library 9, 305
  corpora 306
  data 329
  node in a graph 519
  nodes in a cluster 531
  None value 74
    evaluates to False in conditions
    74
  nonexistent element in a sequence
    104
  nonfatal logic error 55
  nonfatal runtime error 12
  nonsequence collections 138
  normalization 314
  normalized data 471

NoSQL database 370, 503, 517
  column based 517
  Columnar database 517, 518
  Couchbase 517
  CouchDB 518
 =document database 517, 518
  DynamoDB 517
  Google Cloud Datastore 517
  graph database 517, 519
  HBase 517
  key–value 517
  MariaDB ColumnStore 506, 518
  Microsoft Azure Cosmos DB 518
  MongoDB 518
  Redis 517
  not Boolean operator 65, 66, 67
    truth table 67
  not in operator 81, 147
  not_equal universal function (NumPy) 171
  notebook, terminate execution 468
  not-equal-to operator (!=) 41
  noun phrase 309
    extraction 305
  noun_phrases property of a
    TextBlob 309
  null in JSON 224
  number systems
    appendix (online) 193
    binary 193
    hexadecimal 193
    octal 193
  numbers format with their signs (+ 195
  numbers in JSON 224

NumPy (Numerical Python) library
  xix, 8, 130, 136, 160
  add universal function 170
  arange function 164
  array function 161, 162
  broadcasting 168, 171
  convert array to floating-point
    values 472
  full function 163
  hstack function 177
  ones function 164
  multiply universal function
    170
  ones module 160
  ones function 163
  preinstalled in Anaconda 160
  sqrt universal function 170
  statistics functions 130
  transpose rows and columns
    177
  type bool 162
  type float64 162, 163
  type int64 162, 163
  type object 162
  universal functions 170
  unique function 130
  vstack function 177
  zeros function 163

numpy module 130
  random module 166
  choice function 469
  randint function 166
  NVIDIA GPU 463
  NVIDIA Volta Tensor Cores 466

OAuth 2.0 337
  OAuth dance 337
  OAuthHandler class (Tweepy) 341
    set_access_token method 341

object 2, 3, 46
  identity 91
  namespace 292
  type 33, 45
  value 45

object-based programming xii
  object-based programming (OBP) 245, 274

object class 249, 269
  __add__ special method 276, 278
  __format__ special method
    261, 263
  __iadd__ special method 279
  __mul__ special method 276

object-based programming (OBP) 245, 274
percent (%) SQL wildcard character 512
performance xxiii, xxviii
performance tuning 4
PermissionsError 227
persistent 218
persistent connection 355
personal assistants 24
Personality Insights service (IBM Watson) 378
personality theory 378
personality traits 378
personalized medicine 24
personalized shopping 24
petabytes (PB) 19
petaflops 20
phishing elimination 24
pickle module 226
picture xix
Pig 533
Pig Latin 533
pip package manager xxxiv
pitch, voice 377
pixel intensity (grayscale) 407
placeholder in a format string 196
Platform as a Service (PaaS) 504
play function of module pydub.playback 393
plot function of the matplotlib.pyplot module 424
plot method of a class DataFrame 321
plot_model function of the tensorflow.keras.utils.vis_utils module 479
pluralizing words 305, 309
PNG (Portable Network Graphics) 260
polarity of Sentiment named tuple 309, 310
pollution reduction 24
polymorphism 245, 274
pooling layer 476
pop method of dictionary built-in type 140
pop method of list 119
pop method of set 152
Popular Python Data-Science Libraries 8
Popularity of Programming Languages (PYPL) Index 2
population 97
population standard deviation 98
population variance 97, 98
Popup class (folium) 366
position number 103
positive sentiment 309
PostgreSQL 506
pow (power) function of module math 83
power universal function (NumPy) 171
%pprint magic 491
precedence 35
precedence not changed by overloading 277
precedence rules 41, 45
precise monetary calculations 61, 63
Decimal type 61, 63
precision in a scikit-learn classification report 416
%precision magic 167, 309
precision medicine 24
predefined word embeddings 495
predicate 511
predict method of a scikit-learn estimator 413, 435
predict method of class Sequential 482
predicted value in simple linear regression 295
predicting best answers to questions 490
cancer survival 24
disease outbreaks 24
student enrollments 24
weather-sensitive product sales 24
prediction accuracy 414
predictive analytics 24
predictive text input 490, 493
prepare data for machine learning 408
preposition 308
presentation type (string formatting) 193
c 193
d 193
e (or E) 194
f 194
integers in binary, octal or hexadecimal number systems 193
pretrained convnet models 498
pretrained deep neural network models 498
pretrained machine learning models 309
preventative medicine 24
preventing disease outbreaks 24
opioid abuse 24
primary key 507, 508, 509, 510
composite 510
principal 63
principal components analysis (PCA) 439, 452
principal diagonal 414
print built-in function 36
privacy laws 502
private attribute 257
data 250
probabilistic classification 467
probability 76, 153
problem solving 58, 59
procedural programming xix
process dictionary keys in sorted order 143
profile dictionary keys 82
prompt 39
proper singular noun 308
proper subset 149
proper superset 149
property getter method 253
name 253
of a class 251, 253
read-only 253
read-write 253
setter method 253
@property decorator 253
@propertynamer.setter decorator 253
Prospector xxiv
protecting the environment 24
pseudorandom numbers 78
pstats module 82
pstdev function (statistics module) 98
pub/sub system 561
channel 562
topic 562
public attribute 257
public domain card images 258
images 263
Index

read_text method of class Path 314
readability 324
readability assessment 324
readability assessment libraries readability library 324
readability-score library 324
readability formulas Dale-Chall 324, 325 Flesch Reading Ease 324, 325 Flesch-Kincaid 324, 325 Gunning Fog 324, 325 Simple Measure of Gobbledygook (SMOG) 324, 325 reader function of the csv module 236 reading sign language 24 readLine method of a file object 227 readLines method of a file object 221 read-only property 253 read-write property 253 definition 253 real part of a complex number 277 real time 16 reasonable value 212 recall in a scikit-learn classification report 416 %recall magic 133 %save magic 134 recognize method of class SpeechToTextV1 387 recommender systems 24, 329 record 219 record key 219 recurrent neural network (RNN) 293, 370, 460, 489, 490 time step 490 recursion recursion step 93 recursive call 93 recursive factorial1 function 94 visualizing 94 recursive call 95 recursive function 99 Redis 517 reduce dimensionality 476, 494 reduce function 95 of the function1s module 124 reduceByKey method of class RDD 547 reducer in Hadoop MapReduce 532, 536 reducing carbon emissions 24 reducing program development time 76 reduction 95, 122, 124 in functional-style programming 48, 68, 124 pandas 179 redundant parentheses 35, 36, 65 refer to an object 46 regplot function of the Seaborn visualization library 298 regression xix, 400 regression line 293, 296, 298 slope 299 regular expression 7, 82, 203, 208 ^ character 208 \ quantifier 206 ( metacharacter 209 ) metacharacter 209 [] character class 205 {n} quantifier 206 (n, m) quantifier 207 * quantifier 206 \ metacharacter 205 \d character class 205 \D character class 205 \S character class 205 \W character class 205 \w character class 205 + quantifier 206 $ metacharacter 209 anchor 208, 209 caret (^) metacharacter 206 character class 205 escape sequence 205 flags keyword argument 208 group method of a match object 210 groups method of a match object 208, 210 IGNORECASE regular expression flag 208 match object 208 metacharacter 205 parentheses metacharacters, ( and ) 209 search pattern 203 validating data 203 regularization 476 reinforcement learning 27 reinventing the wheel 81 relational database 502, 506 relational database management system (RDBMS) 370, 506 release resources 220 'relu' (Rectified Linear Unit) activation function 475 remainder (in arithmetic) 35 remainder operator (%) 33, 35, 36, 45 remainder universal function (NumPy) 171 remove function of the os module 223 remove method of list 118 remove method of set 152 removing whitespace 197 rename function of the os module 223 repeat a string with multiplication 110 repeat keyword argument of FuncAnimation 157 replace method 199 replacement text 59, 60 repr built-in function 254 __repr__ special method of class object 251, 254, 261 reproducibility xxii, xxvi, 417, 430, 462, 542, 544 in Keras 467 Jupyter Notebooks 12 requirements statement 5, 59 compound interest 63 craps dice game 78 reshape method of ndarray 164, 175, 422 resilient distributed dataset (RDD) 542, 546, 555, 556 resize method of ndarray 175 resolve method of class Path 264 resource acquire 220 release 220 resource leak 251 return statement 73 return_counts keyword argument of NumPy unique function 130 reusable componentry 245 reusable software components 3 reuse 4 reverse keyword argument of list method sort 115 reverse method of list 119 reversed built-in function (reversing sequences) 125 rfind string method 199 ride sharing 24 Ridge estimator from sklearn.linear_model 438
right align > (string formatting) 63, 194
right_shift universal function (NumPy) 171
right-to-left evaluation 45
rindex string method 199
risk monitoring and minimization 24
robo advisers 24
robust application 218
rolling a six-sided die 77
rolling two dice 78, 80
Romeo and Juliet 321
round floating-point numbers for output 60
rounding integers 83
Row class from the pyspark.sql module 557
row in a database table 507, 510, 511, 512, 515
row of a two-dimensional list 126
rpartition string method 201
rsplit string method 200
rstrip string method 197
Rule of Entity Integrity 510
Rule of Referential Integrity 510
%run magic 167
running property of class Stream 358
runtime error 12

S
SalariedCommissionEmployee
class 270
sample method of a DataFrame 430
sample of a population 97
sample variance 97
samples (in machine learning) 407, 428
sampling data 211
sarcasm detection 329
%save magic 167
save method of class Map 367
save method of class Sequential 485
scalar 167
scalar value 465
scatter function (Matplotlib) 440
scatter plot 298, 430
scattergram 298
scatterplot function (Seaborn) 424, 431
scientific computing 5
scientific notation 194
scikit-learn (sklearn) machine-learning library 8, 293, 380, 403, 405, 426, 459
estimator (model) 405, 422
fit method of an estimator 412, 434
predict method of an estimator 413, 435
score method of a classification estimator 414
sklearn.linear_model module 421
sklearn.metrics module 414
sklearn.model_selection module 411
sklearn.preprocessing module 408
SciPy six, 8
scipy 295
SciPy (Scientific Python) library 295, 298
linregress function of the stats module 295, 298
scipy.stats module 295
stats module 298
scipy.stats module 295
scope 87, 290
global 87
local 87
score method of a classification estimator in scikit-learn 414
scraping 204
screen_name property of a User (Twitter) 343
script 9, 42
script mode (IPython) 9
script with command-line arguments 135
Seaborn visualization library xx, 8, 25, 128, 130, 152, 153, 155, 157, 158, 430
barplot function 131
heatmap function 416
module 130
pairplot function 447
predefined color palettes 131
regplot function 298
scatterplot function 424, 431
search a sequence 116
search function of the module re 208
search method of class API 347
search pattern (regular expressions) 203
seasonality 293
secondary storage device 218
secrets module 78
secure random numbers 78
security enhancements 24
seed function of module random 76, 78
seed the random-number generator 78
seek method of a file object 221
SELECT SQL keyword 509
selection criteria 511
selection statement 50
self in a method’s parameter list 249
self-driving cars 24, 26
semi-structured data 502, 517, 518
send a message to an object 4
send method of a socket 552
sentence capitalization 197
Sentence class (TextBlob) 307, 310
correct method 313
lemmatize method 314
stem method 314
sentence property of a TextBlob 307, 310
sentiment 309, 349, 377
sentiment analysis xx, 24, 305, 309, 359, 490
sentiment in tweets 332
Sentiment named tuple 309, 310
subjectivity 309, 310
textblob module 309
sentiment property of a TextBlob 309, 310
sentinel-controlled iteration 59
sentinel value 59
separators, thousands 130
sequence 55, 102
+ operator for concatenation 105
concatenate 105
length 103
nonexistent element 104
of bytes 219
of characters 55, 219
of consecutive integers 57
sequence collections 138
sequence type string 192
Sequential class
compile method 480
evaluate method 482
fit method 480
predict method 482
save method 485
tensorflow.keras.models module 473
smart homes 24
smart meters 24
smart thermostats 24
smart traffic control 24
socket
snippet in IPython 9
SnowNLP 328
social analytics 24
social graph 519
social graph analysis 24
socket 551, 552
accept method 554
bind method 554
close method 554
listen method 554
send method 552
socket function 554
socket module 552
socketTextStream method of class StreamingContext 559
softmax activation function 478
Software as a Service (SaaS) 504
software engineering observations xxviii
solid-state drive 218
sort 115
ascending order 115, 146
descending order 115, 116
sort method of a list 115
sort_index method of a pandas DataFrame 187
sort_values method of a pandas DataFrame 188
sorted built-in function 115, 143, 345
sorted function 68
source code 10
SourceForge 245
spacy module
load function 326
spacy NLP library 326
Doc class 326
ents property 327
label_. property of a Span 327
load function of the spacy module 326
similarity method 328
Span class 327
spam detection 24
Span class (spaCy) 327
label_. property 327
text property 327
Spark (Apache) xix, 503, 530
as a Service (SaaS) 504
batches of streaming data 558
checkpointing 558
fault-tolerance in streaming 558
PySpark library 541
Spark SQL 503, 517, 555, 557
query 557
stateful transformations in streaming 558
streaming 503, 542
streaming batch interval 558
table view of a DataFrame 557
SparkConf class from the pyspark module 546
setAppName method 546
setMaster method 546
SparkContext class 546
textFile method 546
sparkline 563
SparkSession class from the pyspark.sql module 555
spatial data analysis 24
special method 249, 276
special methods
__eq__ 282
__init__ 281
__ne__ 282
__repr__ 281
speech recognition 25, 329
speech synthesis 25, 329
Speech Synthesis Markup Language (SSML) 377
speech to text 16
Speech to Text service (IBM Watson) 377, 382, 383, 384,
385, 386, 387, 391
speech-to-text 329
SpeechToTextV1 class
recognize method 387
SpeechToTextV1 class from the
watson_developer_cloud module 385, 387
spell checking 305
spellcheck method of class
Word 313
spelling correction 305
split function of module re 207
method 221
method of string 145
string method 200
splitlines string method 201
sports recruiting and coaching 24
spread 98
Spyder IDE 12
Spyder Integrated Development Environment xxiv
SQL (Structured Query Language) 506, 507, 511, 515
DELETE FROM statement 511,
516
FROM clause 511
GROUP BY clause 511
INNER JOIN clause 511, 514
INSERT INTO statement 511,
515
key word 511
ON clause 514
ORDER BY clause 511, 512, 513
percent (%) wildcard character 512
query 508
SELECT query 509
SET clause 515
SQL on Hadoop 517
UPDATE statement 511
VALUES clause 515
WHERE clause 511
SQLite database management system 506, 507
sqlite3 command (to create a database) 507
sqlite3 module 7, 82, 507
connect function 508
Connection class 508, 514
Sqoop 533
sqrt (square root) function of
d module math 82, 83
sqrt universal function (NumPy)
170, 171
square brackets 180
SRE_Match object 208
stack 119
overflow 95
unwinding 229, 234
standard deviation 46, 166, 180
standard deviation statistic 98
standard error file object 219
standard file objects 219
standard input file object 219
standard input stream 535
standard output file object 219
standard output stream 535
StandardError class of exceptions 280
standardized reusable component 245
Stanford CoreNLP 328
start index of a slice 110
start method of a
StreamingContext 559
startswith string method 199
stateful transformations (Spark streaming) 558
statement 32
statement spread over several lines 44
statements
break 64
continue 64
del 112
for 50, 51, 55, 57
from...import 89
if 50, 51
if...elif...else 50, 54
if...else 50, 52
import 62, 82
import...as 90
nested 77
return 73
while 50, 51, 55
with 220
static bar chart 102
static code analysis tools 286
static visualization 128
statistics
count 46, 68
maximum 46
mean 67
measures of central tendency 67
measures of dispersion 46
measures of dispersion (spread) 97
measures of variability 46, 97
median 67
minimum 46
mode 67
range 46
standard deviation 46, 98
sum 46, 68
variance 46, 97
statistics module 7, 68, 82
mean function 68
median function 68
mode function 68
pstddev function 98
pvariance function 98
stats 298
Statsmodels 8
Status class (Twitter API)
extended_tweet property 343
text property 343
status property of a User (Twitter) 343
status update (Twitter) 337
std method of ndarray 169
stem method of class Sentence 314
stem method of class Word 314
stemming 305, 309, 314
step in a slice 111
step in function range 60
stock market forecasting 24
stop word 323
elimination 305
stop words 317, 328
stop_stream method of the class Stream 393
Storage as a Service (SaaS) 504
Storm 533
str (string) type 45, 62
str attribute of a pandas Series
181, 212
str built-in function 255
__str__ special method of class object
251, 255, 261
straight-line form 35
Stream class
close method 393
read method 393
stop_stream method 393
Stream class (Tweepy) 357, 358
filter method 358
running property 358
Stream class from module pyaudio
393
StreamingContext class
checkpoint method 558
pyspark.streaming module 558
socketTextStream method
559
start method 559
StreamListener class (Tweepy) 355
on_connect method 355, 356
on_delete method 359
on_error method 355
on_limit method 355
on_status method 355, 356
on_warning method 355
stride 474, 476
string built-in type
* string repetition operator 196
+ concatenation operator 196
byte string 393
capitalize method 197
concatenation 40
count method 198
encode as bytes 553
endswith method 199
find method 199
string built-in type (cont.)
format method 195, 550
in JSON 224
index method 199
isdigit method 202
join method 200
lower method 87, 125, 197
lstrip 197
of characters 37
partition method 201
repeat with multiplication 110
replace method 199
rfind method 199
rindex method 199
rpartition method 201
rsplit method 200
rstrip 197
split method 145, 200
splitlines method 201
startswith method 199
strip 197
title method 197
triple quoted 38
upper method 87, 197
string formatting
fill with 0s 195
left align (<) 194
numbers with their signs (+) 195
presentation type 193
right align (>) 194
string module 7, 82
string sequence type 192
strip string method 197
stripping whitespace 197
structured data 502, 517
Structured Query Language (SQL)
502, 506, 507, 511
student performance assessment 24
Style Guide for Python Code
blank lines above and below control statements 58
class names 84, 248
constants 84
docstring for a function 73
naming constants 260
no spaces around = in keyword arguments 56
spaces around binary operators 32, 43
split long line of code 44
suite indentation 44
triple-quoted strings 38
sub function of module re 207
subclass 4, 245
subjectivity of Sentiment named tuple 309, 310
subordinating conjunction 308
subplots function of module matplotlib.pyplot 264
subscribe to messages 561
subscription operator ([ ]) 103, 105
substring 198
subtract universal function (NumPy) 171
subtraction 33, 36
suite 44
indentation 44
suite vs. block 73, 88
sum built-in function 68, 80, 86
sum method of DataFrame 527
sum method of ndarray 169
sum statistic 337
summarizing documents 329
symmetric difference 463
synapt in biology 463
sybase 506
symbols 337
symmetric difference augmented assignment 151
symmetric_difference method of set 150
symmetric_difference_update method of set 151
synapse in biology 463
synapses 464
synchronous 379
synchronous tweet stream 358
synonyms 305, 315, 316
synset (set of synonyms) 315
Synset class, lemmas method 316
synsets property of class Word 315
syntax error 38
SyntaxError 38, 41
synthesize method of class TextToSpeechV1 392
sys module 7, 82, 135
stderr file stream 219
stdin file stream 219
stdout file stream 219
SystemExit exception 279

T

t attribute of a NumPy ndarray 177
t attribute of a pandas DataFrame 187
tab completion 83
tab escape sequence \t 37
tab stop 37
table 126
in a database 506
table view of a Spark DataFrame 557
tables 506
tags property of a TextBlob 308
tail1 method of a DataFrame 238
tan (tangent) function of module math 83
tan universal function (NumPy) 171
target attribute of a Bunch 407, 428
target in a for statement 55
target values (in machine learning) 428
t-distributed Stochastic Neighbor Embedding (t-SNE) 439
telemedicine 24
tensor 160, 465
0D 465
1D 465
2D 466
3D 466
4D 466
5D 466
Tensor Processing Unit (TPU) 467
TensorBoard 481
dashboard 486
TensorBoard class from the tensorflow.keras.callbacks module 488
TensorBoard for neural network visualization 486
TensorFlow 380
TensorFlow deep learning library 8, 458, 468
tensorflow.keras.callbacks module
TensorBoard class 488
tensorflow.keras.datasets module 461, 468, 490
tensorflow.keras.datasets.imdb module 490
get_word_index function 492
tensorflow.keras.datasets.mnist module 468
load_data function 468, 491
tensorflow.keras.layers.module 473, 495
Conv2D class 475
Dense class 477
Embedding class 495
Flatten class 477
MaxPooling2D class 477
tensorflow.keras.layers.embeddings module 495
Dropout class 495
tensorflow.keras.models.module 473
load_model method 485
Sequential class 473
tensorflow.keras.preprocessing.sequence module 493
pad_sequences function 493
tensorflow.keras.utils module 472, 479
terabytes (TB) 18
teraflop 20
Terminal

macOS xxxiv
or shell Linux xxxiv
Terminal window in JupyterLab 543
terminate method of the class
PyAudio 393
terminate notebook execution 468
terrorist attack prevention 24
testing 4
unit test 287
unit testing 287
testing set 411, 434
testmod function of module doctest 287
verbose output 288
verbos classification 329
text file 218, 219
text index 525
text method of Axes (Matplotlib) 131, 133
text property of a spaCy Span 327
text property of a Status (Twitter) 343
text search 525
text simplification 329
text to speech 16
Text to Speech service (IBM Watson) 377, 382, 383, 384
Textacy NLP library 326
textatistic module 324
dict method of the
Textatistic class 324
readability assessment 324
Textatistic class 324
Index

TextBlob 9, 16
TextBlob NLP library 305
BaseBlob class 307
compare TextBlobs to strings 307
correct method of the
TextBlob class 313
detect_language method of
the TextBlob class 311
infection 305
inter-language translation 305
language detection 305
lemmatization 305
n-gram 305
ngrams method of the TextBlob
class 318
noun phrase extraction 305
noun_phrases property of the
TextBlob class 309
parts-of-speech (POS) tagging 305
pluralizing words 305
Sentence class 307, 310
sentence property of the
TextBlob class 307, 310
sentiment analysis 305
Sentiment named tuple 309
sentiment property of the
TextBlob class 309, 310
singularizing words 305
spell checking 305
spelling correction 305
stemming 305
stop word elimination 305
string methods of the TextBlob
class 307
tags property of the TextBlob
class 308
TextBlob class 307
textblob module 307
tokenization 305
translate method of the
TextBlob class 312
Word class 307, 309
word frequencies 305
word_counts dictionary of the
TextBlob class 315
WordList class 307, 309
WordNet antonyms 305
WordNet integration 305, 315
WordNet synonyms 305
WordNet word definitions 305
words property of the TextBlob
class 307
textblob.sentiments module 310
text-classification algorithm 310
textFile method of the
SparkContext class 546
TextRazor 328
textstat library 324
text-to-speech 329
TextToSpeechV1 class
from module
watson_developer_cloud
385, 391, 392
synthesize method 392
the cloud 16
The Jupyter Notebook 13
The Zen of Python 7
Theano 458, 462
Theano deep learning library 8
theft prevention 24
theoretical science 211
third person singular present verb 308
thousands separator 130, 195
thread 234, 546
three-dimensional graph 442
tight_layout method of a
Matplotlib figure 321
tight_layout method of class
Figure 264
Time class 250, 252
time module 7, 82
time series 293, 370
analysis 293
financial applications 293
forecasting 293
Internet of Things (IoT) 293
observations 293
time step in a recurrent neural
network 490
%timeit magic 165
%timeit module 7, 82
%timeit profiling tool xxiii
timeline (Twitter) 344, 346
Titanic disaster dataset 218, 237,
238
title method of a string 197
titles table of books database 508
tkinter module 82
to_categorical function 472
of the
tensorflow.keras.utils
module 472
to_csv method of a DataFrame
323
to_file method of class
wordCloud 238
token 200
tokenization 200, 305
tokenize a string 144, 145, 207
tokens 305
Tone Analyzer service (IBM
Watson) 378
emotion 378
language style 378
social propensities 378
topic in pub/sub systems 562
topic modeling 329
topical xviii
TPU (Tensor Processing Unit) 459,
466, 476
traceback 34, 55, 233
trailing zero 60
train_test_split function from
sklearn.model_selection
411, 434, 494
training accuracy 497
training set 411, 434
translation services
Microsoft Bing Translator 311
transcriptions of audio 377
transfer learning 459, 479, 485,
498
transform method of the PCA
estimator 452
transform method of the TSNE
estimator 440
transforming data 204, 210
translate method of a TextBlob
312
translate method of class
LanguageTranslatorV3 390
translate speech 26
translating text between languages
377
translation 16
translation services 311
transpose rows and columns in a
pandas DataFrame 187
transpose rows and columns of an
ndarray 177
trend spotting 24
trending topics (Twitter) 333, 349
Trends API (Twitter) 334
trends_available method of class
API 350
trends_closest method of class
API 351
trends_place method of class
API 351
trigonometric cosine 83
trigonometric sine 83
trigonometric tangent 83
trigrams 318
triple-quoted string 38
True 50, 51
True Boolean value 41
truediv division operator (/) 33, 34, 36, 45
tru n universal function (NumPy) 171
truncate 34
truth table 65
try clause 228
try statement 228
TSNE estimator
fit method 440
fit_transform method 440
sklearn.manifold module 439
transform method 440
tuple 80, 102, 106
arbitrary argument list 86
one-element 107
tuple built-in function 109
Turtle graphics
turtle module 82
Tweepy library 334, 340
API class 341, 342
Cursor 344
install 340, 354
OAuthHandler class 341
Stream class 357, 358
StreamListener class 355
wait_on_rate_limit 342
wait_on_rate_limit_notify 342
tweepy module 341
tweepy.models.Status object 343
tweepy.models.User object 342, 344
tweet 337
coordinates 338
created_at 337
tentities 337
extended_tweet 337
favorite_count 338
id 338
id_str 338
lang 338
place 338
retweet_count 338
text 338
user 338
tweet object JSON (Twitter) 343
tweet-preprocessor library 353
set_options function 354
Tweets API (Twitter) 334
24-hour clock format 250
Twitter 333
data mining 332
history 333
rate limits 334
Streaming API 354, 355
timeline 344, 346
trending topics 349
Trends API 332
Twitter API 334
access token 336, 341
token secret 336, 341
Accounts and Users API 334
API key 336, 341
user object JSON 342
API secret key 336, 341
app (for managing credentials) 335
app rate limit 334
Authentication API 334
Consumer API keys 336
credentials 336, 341
fire hose 354
rate limit 334, 342
Trends API 334
tweet object JSON 343
Tweets API 334
user object JSON 342
user rate limit 334
Twitter Python libraries
Birdy 340
Python Twitter Tools 340
python-twitter 340
TweetPony 340
TwitterAPI 340
twitter-gobject 340
Tweepy library 334
Twitter search
operators 348
Twitter Trends API 349
Twitter web services 16
Twittersphere 333
Twittersverse 333
two-dimensional list 126
.txt file extension 220
type dependent formatting 193
type function 33
type hint 283
type of an object 45
TypeError 104, 107
types
float 33, 45
int 33, 45
str 45
typing module 282
ClassVar type annotation 282
List type annotation 282, 283

U
UCI ML hand-written digits dataset 406
ufuncs (universal functions in NumPy) 170
unary operator 66
uncaught exception 234
Underfitting 425
underscore
_SQL wildcard character 512
underscore character ( _) 33
understand information in image and video scenes 376
union augmented assignment 151
union method of set 150
unique function (NumPy) 130
return_counts keyword argument 130
unit testing xxii, 7, 82, 287, 287, 288
United States
geographic center 366
univariate time series 293
universal functions (NumPy) 170
add 171
arccos 171
arcsin 171
arctan 171
bitwise_and 171
bitwise_or 171
bitwise_xor 171
ceil 171
cos 171
divide 171
equal 171
exp 171
fabs 171
floor 171
greater 171
greater_equal 171
hypot 171
invert 171
isinf 171
isnan 171
left_shift 171
less 171
less_equal 171
log 171
logical_and 171
logical_or 171
logical_xor 171
maximum 171
minimum 171
multiply 171
universal functions (NumPy) (cont.)
not_equal 171
power 171
remainder 171
right_shift 171
sin 171
sqrt 171
subtract 171
tan 171
trunc 171
ufuncs 170
unpacking an iterable into function arguments 86
validate a first name 205
uppercase characters 202
utility method 202
valid Python identifier 202
user rate limit (Twitter API) 334
UTC (Coordinated Universal Time) 337
update 140
update Anaconda xxxiv
update method of a dictionary 146
update method of set 151
UPDATE SQL statement 511, 515,
updateStateByKey method of a DStream 559
upper method of a string 87, 197
uppercase characters 202
uppercase letter 33
use cases 23
IBM Watson 374
User class (Twitter API)
description property 343
followers_count property 343
friends_count property 343
td property 342
name property 342
screen_name property 343
status property 343
user object JSON (Twitter) 342
user rate limit (Twitter API) 334
user_timeline method of class API 346
UTC (Coordinated Universal Time) 337
utility method 256
V
V’s of big data 22
valid Python identifier 202
validate a first name 205
validate data 252
validating data (regular expressions) 203
validation accuracy 496
validation_data argument to a Keras model’s fit method 494
validation_split argument to a Keras model’s fit method 481, 494
value of an object 45
ValueError 108
ValueError exception 199
values attribute of a pandas Series 181
values method of dictionary 141
VALUES SQL clause 515
var method of ndarray 169
variable refers to an object 46
variable annotations 283
variance 46, 97, 180
variety (in big data) 23
vector 465
velocity (in big data) 22
veracity (in big data) 23
version control tools xx
vertical stacking (ndarray) 177
vertices in a graph 519
video 376
video closed captioning 329
view (shallow copy) 173
view into a dictionary 142
view method of ndarray 173
view object 173
virtual assistants 376
visual product search 24
Visual Recognition service (IBM Watson) 376
Visual Studio Code 12
visualization xix, 210
die rolling 129
dynamic 152
Folium 363
Matplotlib 128
pandas 240
Seaborn 128, 130
visualize the data 430
visualize word frequencies 319, 321
Visualizing 94
visualizing recursion 94
voice cadence 377
voice inflection 377
voice pitch 377
voice recognition 24
voice search 24
VoltDB (NewSQL database) 520
volume (in big data) 22
vstack function (NumPy) 177
W
’w’ file-open mode 220, 226
’w+’ file-open mode 226
wait_on_rate_limit (Tweepy) 342
wait_on_rate_limit_notify (Tweepy) 342
Watson 374
dashboard 375
lite tiers xxvii
Watson (IBM) xix, xxvi
Watson Assistant service 376
Watson Developer Cloud Python SDK xxvi
Watson Developer Cloud Python SDK 375, 381, 385, 394
Watson Discovery service 378
Watson Knowledge Catalog 380
Watson Knowledge Studio 377
Watson Machine Learning service 380
Watson Studio 379
Business Analytics project 380
Data Engineering project 380
Data Science project 380
Deep Learning project 380
Modeler project 380
Standard project 379
Streams Flow project 380
Visual Recognition project 380
watson_developer_cloud module 381, 385
LanguageTranslatorV3 class 385, 390
SpeechToTextV1 class 385, 387
TextToSpeechV1 class 385, 391, 392
WAV (Waveform Audio File Format) 385, 388, 392
.wav file 385
wave module 385, 393
Waze GPS navigation app 24
Weather Forecasting 24
web service 16, 223, 334, 374
endpoint 334
IBM Watson 374
web services 16
web-based dashboard 16
webbrowser module 82
weighted inputs 465
weights in a neural network 465
WHERE SQL clause 511, 513, 515, 516
while statement 50, 51, 55
else clause 64
whitespace 43
removing 197
whitespace character 200, 202
whitespace character class 207
Wikimedia Commons (public domain images, audio and video) 263
Wikipedia 329
wildcard import 89
wildcard specifier (\*) 525
Windows Anaconda Prompt xxxiv
Windows Azure Storage Blob (WASB) 550
with statement 220
as clause 220
WOEID (Yahoo! Where on Earth ID) 350, 351
word character 205
Word class
  correct method 313
  define method 315
  definitions property 315
  get_synsets method 316
  lemmatize method 313
  spellcheck method 313
  stem method 314
  synsets property 315
  textblob module 307, 309
word cloud 319
word definitions 305, 315
word embeddings 495
  GloVe 495
  Word2Vec 495
word frequencies 305, 315
visualizaion 319
word_counts dictionary of a TextBlob 315
Word2Vec word embeddings 495
WordCloud class 321
  fit_words method 323
  generate method 323
  to_file method 323
wordcloud module 321
WordList class 307
  count method 315
  from the textblob module 307, 309
WordNet 315
  antonyms 305
  synonyms 305
  synset 315
  Textblob integration 305
  word definitions 305
  words property of class TextBlob 307
write method of a file object 220
writelines method of a file object 227
writer function of the csv module 235
writerow method of a CSV writer 235
writerows method of a CSV writer 235
X
XML 502, 518
Y
Yahoo! 532
Yahoo! Where on Earth ID (WOEID) 350, 351
YARN (Yet Another Resource Negotiator) 532, 538
yarn command (Hadoop) 538
Z
Zen 7
Zen of Python 7
  import this 7
ZeroDivisionError 34, 227, 229
zeros function (NumPy) 163
zettabytes (ZB) 19
zip built-in function 125, 132
ZooKeeper 533
zoom a map 363