

# Programming for Business Analytics: An Introduction to Python

Beta (1st) Edition

**Dr. Jeffrey D. Wall**  
[jdwall@mtu.edu](mailto:jdwall@mtu.edu)

# Table of Contents

<b>Section I: Programming Foundations</b>	<b>7</b>
<b>Chapter 1:</b>	
<b>Introduction to Business Programming</b>	<b>8</b>
Components of Business Information Systems	8
Programming for Modern Business	10
The citizen developer movement	11
A word of caution about citizen development	13
Using an Integrated Development Environment for Programming	14
Jupyter Notebook for Python Programming	14
Downloading and starting Jupyter Notebook	16
Jupyter Lab for Python Programming	18
Downloading and starting Jupyter Lab	18
Common Information Systems in Organizations	21
Examples of common information technologies	21
Emerging Information Technology in Organizations	22
Artificial intelligence and machine learning	23
Applications of AI and machine learning	24
Artificial neural networks and deep learning	26
Reinforcement learning	29
The Internet of Things	30
Ethical considerations of sensing technologies	31
Robotics	32
Other Emerging Technologies	33
Chapter 1 Assignment	34
<b>Chapter 2:</b>	
<b>Programming Fundamentals</b>	<b>35</b>
Variables in Programming	35
Variables and data types	36
Procedural Programming	38
Writing functions: parameters, logic, and return values	39
Calling a function	40
Ordered vs. named parameters in function calls	41
Using built-in Python functions	41
Variable scope in procedural programs	42
Using variables within strings	43
Object-oriented Programming	44
Objects and classes	45
Class attributes and methods	47
Variable scope in OOP	50
Getter and setter methods	53

Class diagrams	54
Inheritance in OOP	55
Composition in OOP	58
Chapter 2 Assignment	59
<b>Chapter 3:</b>	
<b>Debugging, AI Code Assistance, Software Testing, and Pair Programming</b>	<b>60</b>
Be Methodical and Patient	60
Errors and Exceptions	61
Errors	61
Exceptions	61
Syntax for catching exceptions	63
Propagating exceptions	65
Throwing exceptions on command	66
Creating custom exceptions	67
Debugging	68
Control flow	68
Breakpoints	70
Breakpoints with Jupyter Notebook	70
Breakpoints with Jupyter Lab	72
Using print() to debug	74
AI Code Assistance	75
Code assistance for debugging	76
Code assistance for generating code	76
Testing Software	79
Unit testing in Python with unittest	80
unittest assert methods	82
Pair Programming	84
Chapter 3 Assignment	85
<b>Section II: Data in Business Programming</b>	<b>86</b>
<b>Chapter 4:</b>	
<b>Introduction to Data Structures</b>	<b>87</b>
Foundational Data Structures	87
Arrays	88
Lists	92
Array lists	93
Linked lists	94
Dictionaries	95
Data Structures and Classes/Objects	96
Introduction to Complex Data Structures	100
Data frames in pandas	100
Tree data structures	101
Decision trees for predicting business outcomes	103
Graph data structures	105

Chapter 4 Assignment	107
<b>Chapter 5:</b>	
<b>Data Structures and Flow Control</b>	<b>108</b>
Flow Control	108
Programming optional control flows	108
Programming alternative control flows	110
Looping for repetitive control flow	113
Using for and while loops with data structures	114
Using foreach loops	116
Looping with objects	118
Recursion	123
Chapter 5 Assignment	126
<b>Chapter 6:</b>	
<b>Introduction to Data Analytics</b>	<b>127</b>
Data Wrangling in Python	127
Reading data from csv and Excel files	127
What is a column in a pandas data frame?	129
Viewing data in data frames	129
View subsets of data in a data frame	130
Viewing the data types of columns	131
Finding and fixing incorrect data types and strange values	132
Identifying and correcting outliers	135
Working with lambda functions	142
Lambda function to identify data type issues	143
Identifying and replacing missing values	144
Collecting Stock Data with Python and APIs	146
Using third-party data collection libraries	146
Using a stock data API directly	147
Descriptive Statistics and Visualizations in Python	149
Descriptive statistics	149
Exploring Basic Relationships in Data	151
Scatter plots	152
Plotting time series data	154
Correlation	155
Covariance	157
Chapter 6 Assignment	158
<b>Chapter 7:</b>	
<b>Integrating Databases into a Program</b>	<b>159</b>
What is a Database?	159
Domain Classes and Database Tables	160
Connecting to Databases within Python Programs	161
SQL Statements and Prepared Statements	163
Inserting data into a database table (create)	167

Selecting data from a database (read)	168
Reading SQL data into a pandas data frame	170
Updating data in a database (update)	171
Deleting data from a database (delete)	174
Integrating database connection objects in domain classes	176
A Cautionary Security Note	181
Chapter 7 Assignment	182
<b>Section III: Programming for User Interfaces</b>	<b>183</b>
<b>Chapter 8:</b>	
<b>Improving the Notebook User Interface</b>	<b>184</b>
Hypertext Markup Language (HTML) for Interface Structure	184
HTML elements	185
Common HTML elements	186
HTML table elements	188
HTML element properties	189
HTML page structure	190
Cascading Style Sheets (CSS) for Aesthetic Designs	192
CSS rules and declarations	192
CSS selectors	195
Tag selectors	195
Id selectors	196
Class selectors	197
Selectors with pseudo-classes	198
Implementing CSS Styles	200
Using HTML and CSS in Notebooks	202
Implementing internal style sheets in notebooks	202
Implementing external style sheets in notebooks	204
Implementing HTML in markdown cells	206
Implementing HTML in code cells	207
Formatting Financial Data	208
Chapter 8 Assignment	209
<b>Chapter 9:</b>	
<b>Creating Interactive Notebook Interfaces</b>	<b>210</b>
Event Driven Programming in Notebooks	210
Interactive HTML Elements	211
Form field elements for user interaction	211
Interactive Form Elements in Notebooks	216
Using ipywidgets for interactivity	216
List of useful interactive widgets	216
Triggering events with on_click() listeners	221
Debugging with ipywidgets	224
Utilizing data from event handlers	224
Using the Output widget to store and access data	225

Using classes to store and access data	227
Interactive Form Elements and Databases	230
Creating Containers for Interactive Widgets	233
Changing the children widgets within a container	234
Chapter 9 Assignment	235
<b>Section IV: Advanced Programming Topics</b>	<b>236</b>
<b>Chapter 10:</b>	
<b>Advanced Data Collection and Preprocessing</b>	<b>237</b>
Advanced Data Collection	237
Web scraping to collect data	237
Scraping data within embedded HTML structures	241
Using API's to collect data	242
Connecting to an API	243
Parsing JSON data	245
Parsing financial data from the SEC's API	246
Advanced Data Preprocessing	249
Levels of measurement	249
Converting nominal data to numeric form	250
Converting ordinal data to numeric form	253
Text Preprocessing	253
Normalizing text	254
Tokenizing text	254
Removing stop words	255
Stemming and lemmatizing text	255
Converting text to a data set	256
Text preprocessing in Python	256
Chapter 10 Assignment	260
<b>Chapter 11:</b>	
<b>Introduction to Machine Learning</b>	<b>261</b>
Dimensionality Reduction	261
Principal component analysis	262
An example of PCA	263
Geometric representations of principal components	265
PCA with Python	267
Cluster Analysis	268
Common types of cluster analysis	269
Centroid-based clustering	269
Density-based clustering	269
Distribution-based clustering	270
Which clustering model to use	271
K-means clustering	271
Supervised Classification Models	275
Decision trees	276

Identifying splitting criteria	277
Splitting based on information gain	278
Splitting based on Gini impurity	280
Implementing decision trees with Python	282
Random forests	284
Implementing random forests with Python	284
Confusion matrices	285
Chapter 11 Assignment	287
<b>Chapter 12:</b>	
<b>Code Repositories</b>	<b>288</b>
Code Repositories with Git	289
Git and the command line	289
Creating/initializing a git repository	291
Checking the status of a repository	291
Adding files and folders to a repository	292
Committing changes to a repository	292
Local versus remote repositories	293
Continuous Integration through Git and GitHub	293
Creating a remote repository	293
Branches and workflows	296
Pushing to and pulling from a remote repository	298
Pull requests and merging branches	299
Chapter 12 Assignment	303
<b>Programming Syntax Cheat Sheet</b>	<b>304</b>
Working with Variables	304
Variable declarations	304
Variable assignment	304
Variable initialization	304
Performing basic mathematical operations with variables	305
Using variables within strings	305
Working with Functions	306
Creating a function	306
Using a function	307
Calling functions with named parameters	307
Working with Classes and Objects	307
Creating a class	308
Using a class to create objects	308
Calling class attributes and methods on an object	309



# Section I: Programming Foundations

This section of the text is dedicated to explaining foundational concepts of programming. Section I includes Chapters 1, 2, and 3.

**Chapter 1** will introduce you broadly to the usefulness of programming in business settings and how you might use your knowledge of programming in the future. Chapter 1 will also introduce you to the various components of information systems used within organizations to enhance business strategy and operations. You will explore existing and emerging information technologies that are changing and will continue to change business practices. You will also be introduced to integrated development environments (IDEs) in Chapter 1.

**Chapter 2** will introduce you to the basic syntax of Python and to foundational concepts of programming, such as variables. Chapter 2 will introduce you to two dominant programming paradigms, namely procedural programming and object-oriented programming. You will learn how to use functions and classes to create functionality within programs. Chapter 2 provides an overview of many topics. Be patient with yourself. You do not need to fully absorb everything from Chapter 2 immediately. Future chapters build on Chapter 2. Over time, your ability to write Python syntax and create procedural and object-oriented programs will improve.

**Chapter 3** will introduce you to foundational ideas related to finding and correcting errors in your programs. Even experts make mistakes when programming. Debugging (i.e., fixing errors in code) takes time and patience whether you are a novice or expert. Don't expect to write error free code or that debugging will necessarily come easily. Chapter 3 will teach you to use debugging tools like breakpoints to help you find errors in your code. This chapter also introduces you to using AI tools to assist you in debugging code and even writing code. Code testing will also be introduced to help you ways to automate quality assurance checks of software.

As you embark on the journey through Section I, please be patient with yourself and with the ideas that will be presented. These chapters won't make you an expert programmer. Absorb as much as you can from these chapters, but recognize that the foundational ideas will be repeated in future chapters. The assignments in the associated workbook will help you practice what you read.

# Chapter 1:

## Introduction to Business Programming

In this text, you will learn how to write computer programs in Python for business purposes. **Python** is a popular programming language with several features that will provide exposure to various programming topics and concepts.

Programming is a highly sought after skill that can **boost your job prospects and your starting salary**, even if you are not a software developer. Software supports business strategy and operations. Understanding programming principles will help you better manage software projects and products in business settings. You should be able to speak intelligently to software developers even if you don't want to be one yourself. Software developers create the software tools (i.e., information systems) used extensively by business professionals. You need to be able to speak with developers to receive the tools that will help you achieve your business objectives. Being able to translate technical lingo into business lingo will also make you a valuable asset among your business peers.

The typical programming text is written for students completing computer science degrees. These texts often focus on nuanced details about programming and programming languages, such as the history of specific programming languages, intricate details about different programming paradigms, details about binary, compilers, and more. This text is not set up in this traditional way. Many of the aforementioned details hold little value to business and analytics professionals, unless you wish to become a full-time software developer. And you may not want that at all.

Instead, this text focuses on teaching programming principles that are directly relevant to business professionals who want to participate in the selection and design of the information systems they use to complete their work. As much as possible, this text provides an applied approach to programming with a focus on developing business analytics applications and business information systems. If you fall in love with programming, you should certainly explore other aspects of programming covered in traditional computer science texts.

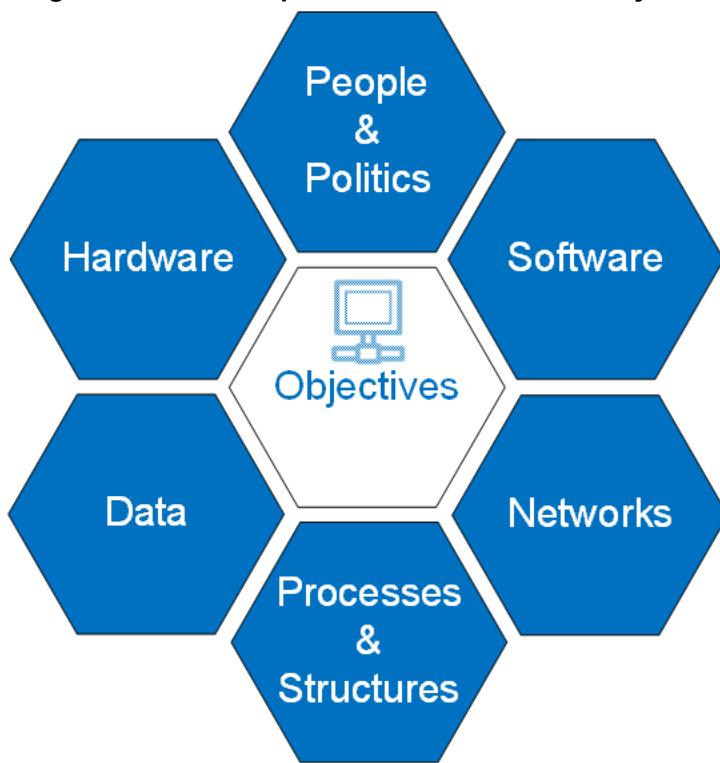
## Components of Business Information Systems

Information systems are more than just software programs written in a programming language. Information systems consist of hardware, software, data, business processes, organizational structures, business goals and objectives, people, and organizational politics. Figure 1.1 shows the various components of an information system. You will learn more about how these different components work together to form an information system in other texts. This text will primarily focus on the software portion of information systems.

Yet, one cannot develop valuable software without understanding how some of these other non-software components can and should influence software design. **Business goals and objectives** should drive the decision to start and continue software projects and products. Developing information systems should not be a goal or objective in itself. Too often, IT professionals look to software to remedy business problems without considering other alternatives.

For example, organizations can simply purchase software from vendors rather than build software in-house. Other times, a simple change to a manual business process may be all that is needed to solve problems, improve performance, and/or reduce employee or client frustrations. Software applications aren't always needed to solve business problems. Always consider alternatives to building software before taking on the costs of software development. Software development is expensive and requires IT staff to support and maintain the software and hardware, train users, and more. Decisions to build software should be scrutinized according to the costs and benefits of developing a proprietary software system as compared to other solutions that can achieve business goals and objectives. The value of software is dependent on context.

**Figure 1.1: The Components of an Information System**



You must also consider **people, politics, processes, and organizational structures** before building a business software system. Organizations are complex and so are people. To begin a software project, whether building or buying a software system, you should ensure that you understand who will directly use the software and who will indirectly be affected by the software. As a simple example, consider developing a specialized customer relationship management (CRM) system for a company. A CRM system keeps track of customers and their experiences with the company. The marketing and customer service departments will certainly take an interest in the CRM software. But who else will the system affect? Even if customers do not directly use the CRM system, they will be affected indirectly by potential changes in customer service workflows built into the CRM software. Further, any changes to the way data is collected about customers could affect other systems and processes within the organization that rely on customer data. Be prepared to carefully think about who might be affected by the programs you build or buy. Seek diverse user and stakeholder input as you develop or purchase software.

You must also learn whether political support exists for the software product you wish to build or buy. **Key users and stakeholders** ultimately determine the success of an information system. Sometimes management will request a software build or purchase without considering the perspectives of the users

who will ultimately use it. For example, hospital administrators have implemented information systems within hospitals that were not liked by doctors and nurses. The investment in these systems was wasted because the systems went unused. This problem exists across industries and organizations. The mantra, "if you build it they will come" does not often hold true in the software industry.

If you know the values and norms of key users, and tailor information systems to support those values and norms, you will have greater success in developing or purchasing valuable software. However, there will be times that business strategy requires a change among your users' values and norms, particularly when the users are employees of your company. In such cases, finding a strong **sponsor** (a high-level executive or manager) to promote change is essential. A project or product sponsor can help to convince users (i.e., other employees and departments) to change perspectives and adopt new business software systems with the associated norms and values embedded in the systems. Learn about the people, processes, structures, and politics that surround systems you are asked to build or buy.

IT and business teams should also consider how a new software system will work with the many other software systems that will surely exist within the organization. **Interoperability** refers to how well different software systems interact with one another. Learn the technology environment you will work in. Learn how different systems are interconnected. This will take time, so don't be discouraged if the task seems insurmountable when you first join an organization. Organizations have many different systems that work together (or sometimes fail to work together) to support the organization's mission. You will learn the architecture of the systems within your organization over time as needed. Development teams also need to consider where their software will run. Software runs on **hardware** and utilizes **networks** to pass messages from one software system to another and from software to user.

As you learn to write code, remember to keep all of the components of an information system in mind. An information system is far more than just software code. If you remember that principle, your organization can develop or purchase more valuable software, and you can become a manager leading in an increasingly technical work environment. As a business professional, you may be called upon to help software developers build new systems that account for people, politics, and processes within your organization. Knowing how software is written will help you work with software developers even if you don't intend to write code yourself. The future of work will be software assisted or automated. Prepare to live in that world.

## Programming for Modern Business

Computer programs touch every aspect of business. Programming is used by many companies to: invest resources wisely; track assets and liabilities; market products; find, track, and retain customers; find and manage employees; and to generally make better and/or faster business operational and strategic decisions. In particular, business analytics systems are designed to help managers and employees make wise decisions. Analytics systems seek to collect, organize, and process information to extract valuable business insight. This text will help you explore aspects of analytics systems that pertain to business users.

The burgeoning use of technology to support business operations, in conjunction with recent information technology advancements, will soon impact what jobs exist and how you will complete your work. The last industrial revolution led to the automation of many manual labor jobs. The ubiquity of information systems and the use of increasingly intelligent information technology is leading to another industrial revolution and a new wave of automation. This wave of automation is fueled by advances in artificial intelligence, machine learning, the Internet of things, blockchain technology, and robotics. Quantum computing will likely play an important role in this revolution as well. Although past revolutions primarily led to the automation of manual labor, the new wave of automation is also affecting and will continue to affect

information work, such as work completed by managers, business analysts, accountants, financial analysts, software developers, etc.

As someone who is interested in the future of work, you should be concerned with understanding automation technologies and how to become a leader in this next wave of computerized automation. You don't necessarily need to understand the intricacies of programming, machine learning, large language models like ChatGPT, or other technologies to meaningfully prepare yourself for the future (though it can help if you have the inclination). You do need to understand how these technologies are affecting and will affect business strategy, processes, and practices. Keeping track of advances in information technology will prepare you to adopt the right technologies at the right time in your future organization. Some technologies are too novel to adopt without major IT labor and infrastructure expenses. However, late adoption of certain technologies can cause a company to become obsolete.

Unfortunately, business information systems and analytics systems are sometimes overhyped. It isn't easy to identify what technologies you should invest in and when. Although many great things can be accomplished with computing and analytics, it is important to recognize that software can be expensive and time consuming to develop. Truly expert and creative programmers and analytics professionals are expensive and hard to find and retain. IT managers have their hands full keeping IT talent contented and challenged. It can also be expensive to purchase pre-built software (i.e., vendor solutions). Further, information systems, particularly those that are emerging within an industry, may not produce the overhyped marketing results promised. Still, to remain competitive and compliant with regulations, many organizations must purchase or build software and stay abreast of the latest advances in computing.

One problem with software development is that domain experts (i.e., engineers, marketing professionals, financial experts, accounting experts, etc.) don't always play a crucial enough role in the development of the systems. By learning to program and by developing stronger analytics skills, you will be able to serve as a bridge between technical experts (i.e., software developers and data scientists) and domain experts (i.e., managers, accountants, engineers, marketing professionals, etc.). You will be able to explain business problems to software developers and data scientists in ways that make sense to them. You will also better understand the feasibility of utilizing certain technologies to enhance business strategy and improve business operations. Additionally, you will be better equipped to support business peers as you explain technical concepts to them in "friendly" ways.

This chapter will explore some existing and emerging technologies to help you understand the dynamic and changing nature of information work within organizations. We will start with a discussion of the citizen developer movement, move to some existing information technologies widely used in industry, and then explore some emerging information technologies that will change the nature of information work.

## ***The citizen developer movement***

Designing and developing computer programs and larger information systems has largely fallen to IT departments within organizations. Traditionally, IT departments have been responsible for either purchasing software from vendors or developing proprietary software for business users. Under this paradigm, business users had relatively little to do with the design and development of software, beyond identifying system requirements (i.e., the needed functionality of the particular system).

Historically, the IT labor force has been expensive to manage, because IT skills are in high demand and in relatively low supply. Organizations do not have unlimited resources. This often leaves IT departments

with a plethora of work to do and few people to complete the work. IT departments often have backlogs of work that business users would like the IT department to complete. This plethora of work requires IT departments to use IT labor resources and information technology strategically. IT resources are often dedicated to support large, strategic initiatives and key business operations. This means that many tedious tasks that could be easily automated or improved may go unchanged. However, recent changes in the technical skills of the workforce are altering this paradigm.

Increasingly, business professionals, engineers, and other non-IT domain experts enter the workforce with basic computing knowledge. For example, you will possess rudimentary programming skills after reading this text and completing the associated programming exercises. As tech-savvy business users enter the workforce, they are able to find or develop programs and systems to make their own work easier to accomplish. Tech-savvy business users are able to automate tedious, non-strategic tasks that IT departments do not have time nor resources to address. In automating these simple and tedious tasks, business professionals free themselves to work on more interesting and valuable work tasks. The movement in which non-IT employees assist in the development or purchase of software systems is referred to as the “**citizen developer movement**.” In this movement, “citizens” are non-IT employees who possess technical knowledge that allows them to build or find and use simple (and sometimes even complex) information systems.

With further practice and use of the technical knowledge and skills you acquire, you can participate in the citizen developer movement. As you enter the workforce, you will encounter many information technologies that will assist you in your work, but you will still find that you do many repetitive and menial tasks. Pay attention to your work and ask yourself if some of your work tasks can be automated with simple computing logic. If it can be automated, you might be able to develop a simple program or system to automate tedious and boring tasks to free you to work on more interesting and valuable tasks.

How can you automate your work? You will find many technologies to support you in this effort. The high-level programming language you will learn in this text (i.e., Python) is fairly simple to use compared to lower-level programming languages like the C programming language or assembly languages. You will learn how to use the basic logic and components of many systems, such as databases, flow control logic, objects, and functions. However, you don’t always need these raw coding skills to participate in the citizen developer movement.

You are probably familiar with spreadsheet technologies, such as Excel. Spreadsheets can be automated through the creation of macros and through simple programming scripts. Be cautious with macros though. They are not considered particularly secure. You can create a simple user interface in Excel using spreadsheet cells. In Excel, you can write programming logic in a language called Visual Basic for Applications (VBA) to automatically perform calculations as new data is entered into a spreadsheet. Although the syntax is slightly different than Python, VBA still relies on programming logic, such as conditional logic, loops, variables, methods, etc. to produce outputs given a set of inputs. You will learn about these different programming concepts in later chapters. Excel even has plugins that allow you to write Python to automate spreadsheets. However, spreadsheet automation lacks the robustness and security protections afforded by other types of systems. Spreadsheet automation can be used as a stepping stone toward automation, but be careful in overusing spreadsheets as information systems.

An emerging technology called **no-code and low-code platforms** also allow business users to create programs, sometimes without writing any code at all. No-code and low-code platforms are gaining popularity in industry among business users and within under-resourced IT departments. No-code and

low-code platforms provide a graphical user interface to help business professionals set up a database and create interfaces for users to enter business information and view reports. These systems often possess built-in functions that you can call, much like the built-in functions in Excel. Although no-code and low-code platforms tend to have better security than spreadsheets and are more robust and systematic, they still pose security issues when compared to traditional systems. No-code and low-code systems are still developing capabilities to support large enterprises. So be cautious when considering no-code and low-code platforms to support large enterprises. Still, you should keep your eye on these technologies and explore how they can make your work simpler.

Large language models, such as ChatGPT, that generate an understanding of human language also provide tools to support citizen developers. Many large language models like ChatGPT, Google's Bard, and Microsoft's Bing AI Chatbot all provide some ability to generate code based on textual descriptions of what you would like. Other coding-specific tools, such as GitHub's Copilot, are designed to support developers in improving their coding productivity. Although many of these tools currently require a basic understanding of programming concepts, for those who possess this knowledge, they can free you from the tedium of writing code. This book will provide you with the foundational programming concepts that can help you write functioning code using some of these AI tools to assist you. It is likely that future improvements in large language models or other forms of AI will lead to even simpler tools that require less and less knowledge of programming concepts.

### **A word of caution about citizen development**

Although some praise the citizen developer movement for its ability to solve the low-level problems that IT departments don't have time and resources to tackle, others (often within IT departments) view the citizen developer movement as dangerous. Although you will have budding programming skills after reading this text, you will still lack much knowledge about how to write programs that are secure, consistent, and free from errors. Even students who graduate college with computer science degrees are novices that require experience to refine their knowledge. They require training and support from senior developers to learn to develop secure and high-quality systems. IT departments rely on many processes to ensure that systems are developed in a secure and high-quality manner. The systems and programs developed or downloaded by citizen developers can be seen as a threat to these carefully planned and executed efforts. Sometimes these citizen developed systems are called "shadow systems" because they operate in the shadows, outside of the purview of the IT department.

If you wish to engage in citizen development work within your organization, be sure to check with your IT department first. Many IT departments are developing policies, procedures, and training to help citizen developers produce secure and viable systems. In some organizations, IT employees and business users work together to build secure citizen-developed systems. Be cautious as you use the technical skills you will learn in this text. Don't assume you know more than you actually do.

Organizations that support citizen development should produce learning resources and guides to help employees use technology to automate small tasks in a secure and consistent manner. Look to these resources for guidance. For example, large financial firms that are trying to keep pace with technology advancements in the finance industry are encouraging large portions of their labor force to learn programming through training offered by their IT departments. Engaging in citizen development can set you apart from other employees in your organization. You will be more likely to be seen as someone capable of leading automation efforts. As more and more work is automated, you want to be seen as someone who can adapt to these changes and even lead them.

If your organization doesn't provide programs or guidance for citizen developers, you may be able to help establish these training programs. Establishing a citizen developer support program offers another way to set yourself apart in an increasingly technical and rapidly changing business environment. Working with the IT department, you can develop training and policy to support citizen development. Be creative and always keep the future of work in mind. In many ways, you can either be an architect of the future of work or become an "object" affected by the changes others are architecting. Choose carefully.

## Using an Integrated Development Environment for Programming

Although it is possible to write programming code in a simple text editor (e.g., Notepad), it can be extremely painful to do so. Text editors do not help you identify simple errors in your code, nor do they offer autocomplete suggestions to help you program quickly. Simple text editors also lack tools to help you compile (i.e., convert) your code to something a computer can understand. These simple editors do not even possess a way to view the output of your code. Although simple text editors do not provide such features, **integrated development environments (IDEs)** offer at least some of these features, and often more. Some IDEs even provide support for AI-assisted code writing.

In this text, we will introduce the **Jupyter Notebook IDE** and **Jupyter Lab IDE** for writing analytics programs in Python. Jupyter Notebook and Jupyter Lab are simple IDEs, but they possess several desirable features for business analytics purposes. Other programs, such as business information systems written in Java or Go (and even Python), are typically built with more feature-rich IDEs. For our purposes, many of these advanced IDE features are not necessary and can even cause confusion for a first time programmer. The Jupyter IDEs will keep things simple and help you focus on the foundations of programming and less on the specific features of a particular IDE. They will also prepare you to write analytics programs to support analyses you conduct as a business professional.

### ***Jupyter Notebook for Python Programming***

Jupyter Notebook is an IDE used primarily by analytics and data science professionals. Jupyter Notebook runs inside of a browser, such as Chrome or Firefox. Although notebooks can be created for many programming languages, Python is the default language associated with Jupyter Notebook. Jupyter Notebook is founded on the idea of **notebook documents** (a.k.a., notebooks). When writing code in Python with Jupyter Notebook, you write text and code in a notebook and directly share the notebook with your users. What you write in a notebook is what others will see. The notebook is both the code and the user interface that displays analytics results to managers and peers interested in your work.

The Jupyter IDEs (i.e., Jupyter Notebook and Jupyter Lab) are used primarily for exploration of datasets and experimentation with statistical models. They are not particularly strong IDEs for writing production ready code used in business information systems. What this means is that code you write in the Jupyter IDEs will eventually be moved to another IDE, such as PyCharm or Visual Studio, that is better equipped to write traditional user-facing systems. However, this assumes you are creating an analytics system that will be used repeatedly by multiple users across the organization. If you are just analyzing a dataset to answer a specific question, then the Jupyter IDEs can be ideal. They can also be ideal for writing the first iteration of a program. Programming code can then be transferred to more complex IDEs and product development processes.

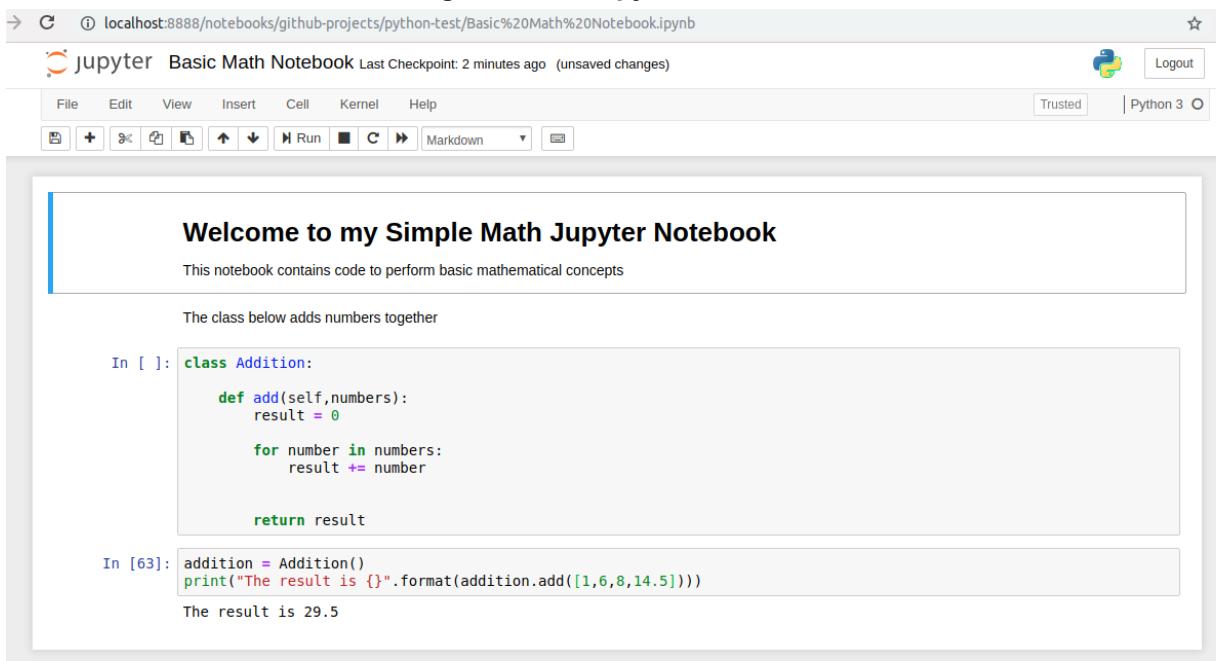
Notebooks consist of multiple **cells** where text or code can be written. You must choose what each cell contains (e.g., text or code). Cells that contain code (i.e., code cells) can be run (i.e., executed) to produce outputs based on calculations and programming commands, such as the results of statistical analyses or graphs and other data visualizations. Cells that contain text (i.e., markdown cells) can be used to document decisions you make in your code cells. Notebooks are useful for coding, but also for explaining code to non-technical users or to other programmers. Business analytics and machine learning models are complex and require careful experimentation and testing. Notebooks can help with this early exploratory process.

Notebooks are useful for data exploration because each cell runs individually from other cells. After running a cell, the information in that cell remains in computer memory to be used by other cells. This feature is helpful when you need to import and analyze a large data file, such as a large spreadsheet or database. Because each cell runs independently, you can place the data import logic in one cell and the data analysis logic in other cells. By doing so, you only need to run the data import cell once. Importing data takes time for large files and you want to minimize your wait time when programming. You can make multiple changes to the data analysis cells and re-run those cells without needing to re-import the data each time you change the analysis logic. Other IDEs do not provide this cell-based, notebook functionality. This is one of the key advantages of the Jupyter IDEs and other notebook-based IDEs.

Notebooks are also useful for analytics purposes because a notebook can be used to document all of the decisions you make about how to handle data, the assumptions you make when choosing particular statistical and mathematical models, and the interpretations you make about the results of statistical and mathematical models. This textual documentation is contained alongside your actual code. **Notebooks make it easy to share your analytic decisions with others to receive feedback.** The notebooks can also be used to present data similar to slideshows (e.g., Power Point Slideshows). Figure 1.2 shows what a notebook looks like.

Notice that the first two cells contain text welcoming you and telling you what the code does. The next two cells contain Python code. In the last cell, you can see the output of the cell calculations beneath the cell. Don't worry about the Python syntax in the example just yet. For now, pay attention to what the notebook and cells look like.

**Figure 1.2: A Jupyter Notebook**



Jupyter Notebook is a rather simplistic IDE. Although other more complex IDEs exist for writing Python code, such as Spyder or PyCharm, this text focuses on business analytics programming. As such, a simple notebook IDE is sufficient. Jupyter Notebook is commonly used by business professionals for writing analytics programs. Let's see how to use the basic features of a Jupyter Notebook.

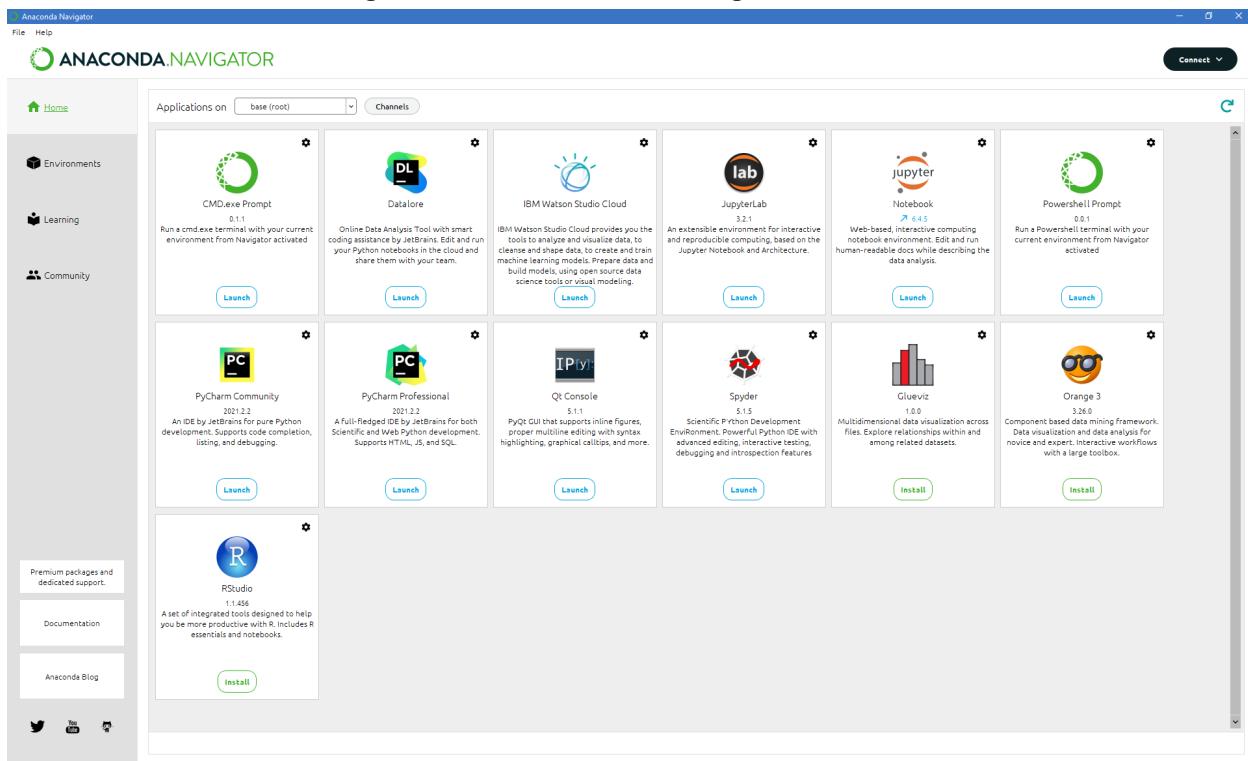
### **Downloading and starting Jupyter Notebook**

The **easiest way to download Jupyter Notebook** is by downloading Anaconda. **Anaconda** is a data science platform that comes bundled with Python, Jupyter Notebook, Jupyter Lab, and several commonly used Python analytics libraries. It is possible to download each of these components separately, but it will be time consuming and is unnecessary. **Make sure to download and use the Python 3 version of Anaconda** and NOT the Python 2 version. Python 3 is the latest version of Python at the time of this writing. For simplicity, download Jupyter Notebook through Anaconda at <https://www.anaconda.com/distribution/>.

Anaconda is a large program, so don't worry if it takes a long time to install. For the purposes of this class, you can choose all of the default selected installation options unless you wish to change them.

**If you download Jupyter Notebook through Anaconda**, start the **Anaconda Navigator** and then choose "Launch" under the Jupyter Notebook logo. After clicking the launch button, Jupyter Notebook will open in a browser. Figure 1.3 shows the Anaconda Navigator interface.

**Figure 1.3: The Anaconda Navigator Interface**



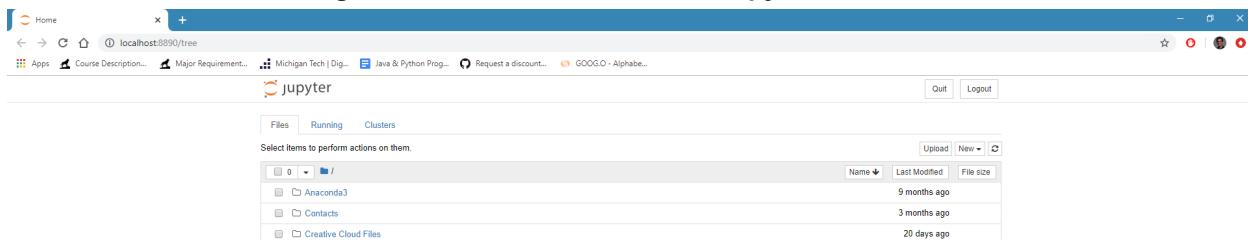
**If you install Jupyter Notebook as a separate program** and not through Anaconda, open the Terminal in Linux/Mac, or Command Prompt in Windows. Then type the following command and press Enter:

```
jupyter notebook
```

After pressing enter, the command line will open a browser tab with a working copy of Jupyter Notebook. If you run Jupyter Notebook as a standalone program, leave the terminal/command line window open when you are working in Jupyter Notebook. If you close the terminal/command line, it may close your notebook or make it non-functional.

Regardless of whether you use Anaconda or a stand-alone installation of Jupyter Notebook, the first page you will encounter in the browser shows the file structure of your computer. Figure 1.4 shows the starting page of Jupyter Notebook.

**Figure 1.4: The Start Window of Jupyter Notebook**



If you have already created a notebook, you can use the folders in the start window to browse to the location where your notebook file is saved and click on the file to open it. If you are creating a new notebook, you can browse to the location where you wish to save your notebook file. It is often easier to create new folders within your filesystem (e.g., Windows Explorer) rather than using the Jupyter Notebook IDE. However, if you wish to create the folder with Jupyter Notebook, click the "New" dropdown on the right hand side of the screen and select "Folder". You can rename the folder by clicking the checkbox next to the newly created "Untitled Folder" and clicking the "Rename" button that appears after checking the box.

Once you navigate to the folder where you want to save your notebook, click the "New" dropdown again and **choose "Python 3"**. This will open a new tab in your browser with a blank notebook. Figure 1.2 shows the notebook window.

Once the notebook window is open, the controls are fairly simple. You can **save your progress** with the save icon, you can **add new cells** to the notebook with the + icon, and you can **run the code within cells** using the Run button. For each cell that you create within the notebook, you must **set the type of information the cell contains** using the dropdown that says "Code" by default. You can make a cell accept Python code by ensuring that "Code" is selected in the dropdown. If you wish to write text to describe what your program is doing, you can select "Markdown" instead of "Code". If you know how to use HTML, you can add HTML into "Markdown" cells in addition to text. Later chapters will introduce HTML to help you structure the output of your cells.

These basic controls in Jupyter Notebook will help you create analytics programs. Jupyter Notebook is a simple IDE, which can save you time in learning how to control the IDE, but it can also feel a bit clunky at times.

## **Jupyter Lab for Python Programming**

Jupyter Lab is a slightly more advanced notebook-based IDE that possesses a few more features to locate errors in your code. It also offers usability improvements to the Jupyter Notebook interface. You may prefer the Jupyter Lab IDE over the Jupyter Notebook IDE. However, both IDEs can be used to write analytics programs.

The core functionality of Jupyter Lab is the same as in Jupyter Notebook. Jupyter Lab still uses cells to divide computing logic into chunks that run individually. Be sure to separate your computing logic so that code that takes a long time to execute is separated from code that changes regularly and runs quickly. Business analytics and machine learning programs often require time-intensive computing tasks. Don't try to put all of your logic into one cell. It will work, but that practice makes editing and testing your code time consuming. Take advantage of the cell-based features provided by notebooks.

## **Downloading and starting Jupyter Lab**

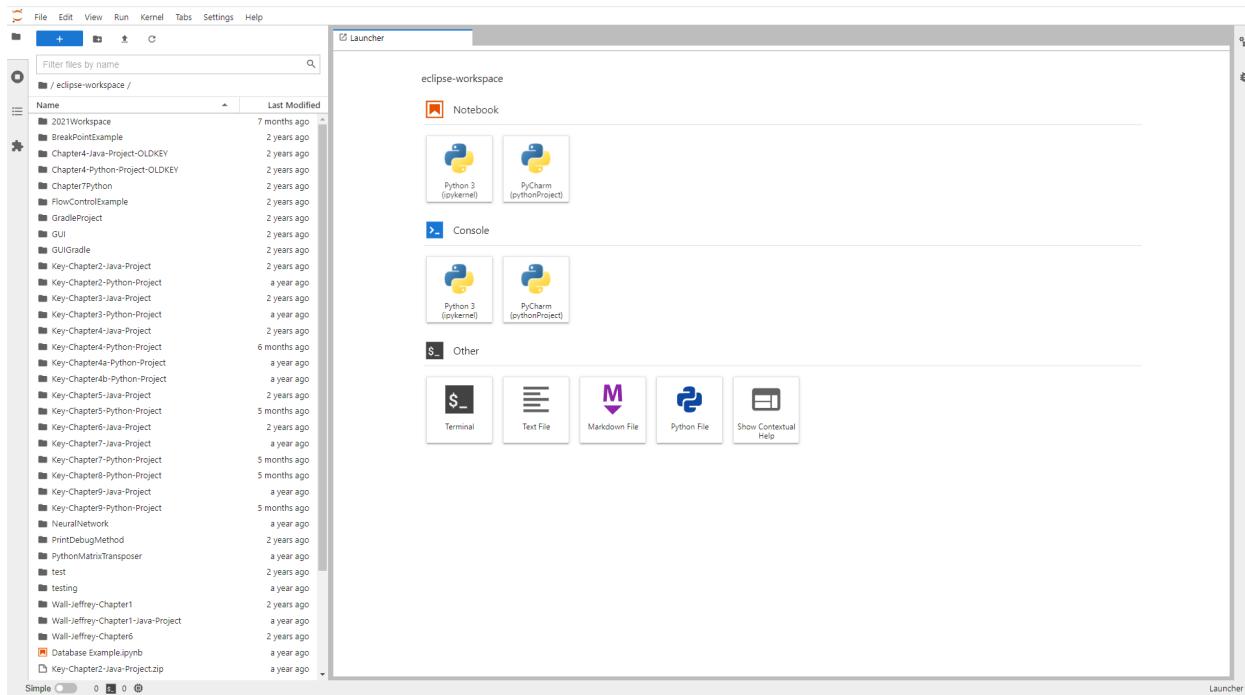
Jupyter Lab also comes with Anaconda. For this reason, it is strongly advised that you download Jupyter Notebook and Jupyter Lab through Anaconda. Again, it is possible to install Jupyter Lab as a standalone program, but this will require a little more computing knowledge. When you open the Anaconda Navigator, you will see Jupyter Lab as one of the options to install/launch. If it does not come pre-installed, simply click the "install" button under the Jupyter Lab logo and Anaconda will install Jupyter Lab.

To start Jupyter Lab, simply click the “Launch” button under the Jupyter Lab logo within Anaconda Navigator. Like Jupyter Notebook, Jupyter Lab will open in a browser window. Unlike Jupyter Notebook, the Jupyter Lab interface doesn’t have a separate start window. The entire interface exists on one browser page. All operations can be completed through the different sections in the IDE interface. Figure 1.5 shows the Jupyter Lab interface.

Notice that the left side region provides a way to view your filesystem so you can access and manage your notebook files. The main part of the interface is the “Launcher”. The launcher allows you to create new notebooks. Multiple notebooks can be opened simultaneously. Each notebook will appear in a different tab within the Launcher section of the interface.

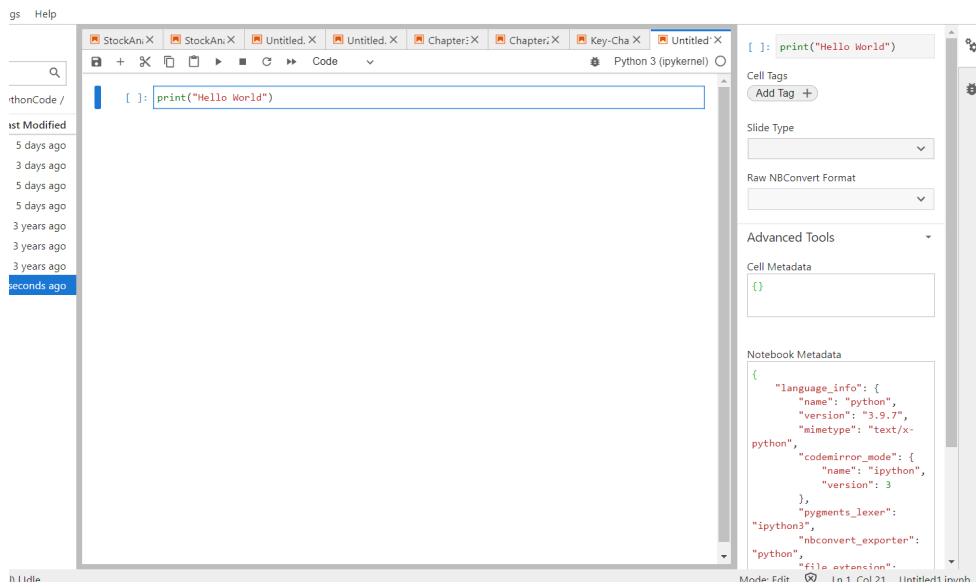
Before creating a new notebook, be sure that you use the left sidebar to navigate to the location where you want to save the notebook. Once you select the correct file folder, simply click the “Python 3” icon in the main launcher window and a new notebook will be created. You will then see the new notebook appear in the left sidebar file explorer area. You can rename the new notebooks using the file explorer navigator in the left sidebar. Simply right click on your newly created notebook file and choose “Rename” from the list of menu options. Excepting some debugging (i.e., error correction) features that will be described in later chapters, Jupyter Lab notebooks rely on the same controls as described in the Jupyter Notebook section (e.g., save and run). Now that we have introduced you to how to write analytics programs in an IDE, we will now turn to some common information systems used in organizations.

**Figure 1.5: The Jupyter Lab Interface**



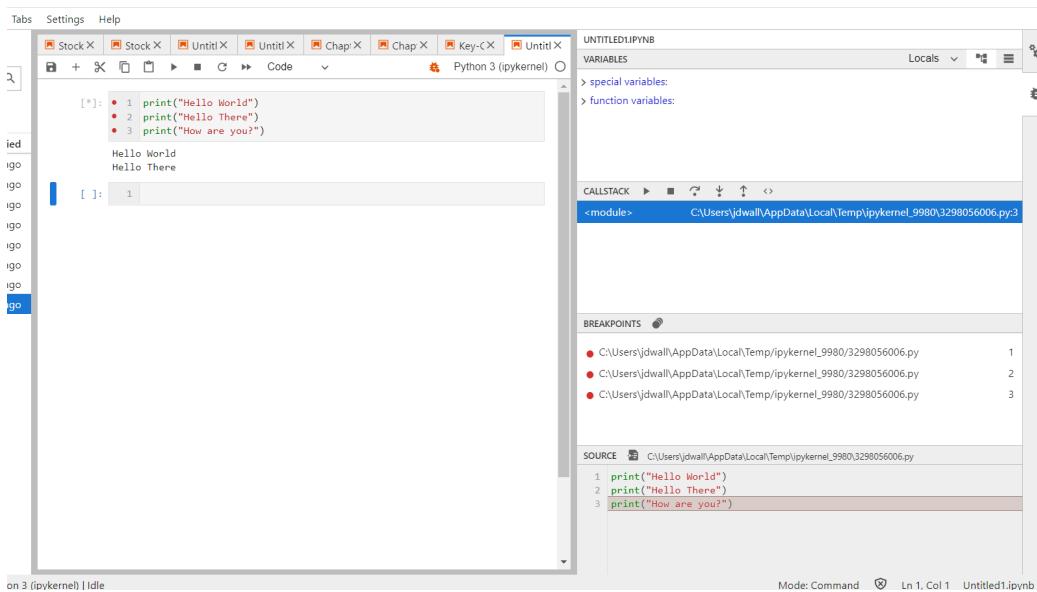
On the right hand side of the page, you will notice a couple of icons: a gear icon and an icon that resembles a beetle. If you click the gear icon, you will see metadata tools for the selected cell in the open notebook. Figure 1.6 shows the expanded metadata window. We will not use this particular window in this text. This window and its tools are for advanced notebook manipulation.

**Figure 1.6: Metadata Tools for Advanced Cell Manipulation**



The beetle icon will open the debugging window. The debugging window has tools to help you find and correct errors in your code. You will learn more about how to use the debugging window in a later chapter. The Variables section of the debugging window allows you to analyze the values of variables at different points in the program. The Callstack section shows the command that is currently being executed by the Python interpreter. The Breakpoints section lists any breakpoints used for debugging purposes. You will learn more about breakpoints in later chapters. The Source section shows you the code that is being executed. Figure 1.7 shows the debugging window. You will often want to extend this window by dragging the bar between the debugging window and the main coding window to the left.

**Figure 1.7: The Debugging Window**



## Common Information Systems in Organizations

Organizations have relied on information technology for decades. Business information technology started with simple spreadsheets and has developed into complex systems with multiple interconnected servers (i.e., powerful computers) and hundreds of thousands to millions of lines of code.

Not every organization invests in technology in the same way. Some organizations set themselves apart by revolutionizing business processes and opening new markets through the development of new information technologies. Companies like Google, Oracle, Microsoft, Amazon, NVIDIA, Meta/Facebook, and other technology-centered firms are clearly leading the development of many general use technological innovations, such as database technologies, cloud computing technologies, and chip technologies. However, you will find organizations in every industry that provide technology solutions to other organizations within their specific industry.

For example, Epic supports the healthcare industry with information systems to manage operations specific to hospitals and medical clinics. Many financial firms, such as JP Morgan Chase and Citigroup, are trying to rebrand themselves as technology firms in the finance industry to compete with tech startups in the industry. In the accounting industry, firms trying to invent the future of accounting are looking to blockchain technology and other related technologies to automate continuous audits and other accounting functions. These are only a few examples of organizations leading technological advancement in their fields. These types of organizations require substantial investment in information technology and IT labor. IT departments in these firms require investment in software developers and strong IT-focused managers and technology visionaries. They may also rely on citizen development. A firm trying to become a technology leader may need to train existing non-IT employees to program or at least to think about work in new and technology-centered ways.

Although every industry has technology leaders, many companies within an industry take a different approach. These organizations patiently wait for technology leaders to create and mature information technologies and practices. These organizations then purchase and use the technology produced by the leaders. IT departments in these companies consist of fewer developers and more information systems professionals who can make decisions about what technology to purchase and how third-party systems should be configured for their organizations. In these firms, business users are trained on these third-party technologies. Such organizations use a third-party technology until a better solution comes along. Although this is a common strategy, firms that adopt it still require thoughtful leaders who pay attention to technology advancements. In industrial revolutions, firms that fail to pay attention to advancements in technology can find themselves outdated and unable to compete in a changing industry. The forthcoming industrial revolution will leave many companies without a customer base as new competitors enter and change the market. Be attentive!

### ***Examples of common information technologies***

Whether you join a technology leader like Microsoft or Google, or a company that primarily relies on third-party systems, all companies have similar technological needs. Even Google and Microsoft use third-party technologies to support their basic business needs.

Large organizations commonly rely on software called enterprise resource planning systems (ERP systems) to manage their business operations. There are many ERP system vendors, such as SAP and Oracle. ERP systems contain many customizable modules or subsystems that help executives, managers, and other information workers track important business information to make informed business decisions.

For example, an ERP system has important finance and accounting modules that allow executives and managers to monitor assets and financial resources to make decisions about how resources should be used by the organization. ERP systems also contain modules to monitor customers. These customer-centric modules are called **customer relationship management systems** (CRM systems). CRM systems track the various touchpoints the organization has with its customers. For example, CRM systems track purchases and customer service contact. Another common module is the **supply chain management system** (SCM system). SCM systems help organizations keep track of their supply chain partners that supply the organization with important parts and materials or distribute products to consumers. SCM systems allow organizations in a supply chain to share information to make the supply of goods and services efficient and flow smoothly across the supply chain from raw material providers all the way to the end customer.

You will likely interact with an ERP system if you join a large organization. Traditional ERP systems, such as those designed by SAP, are very large with many modules like those described above (and far more). Oftentimes, organizations don't need all of the modules contained within these large ERP systems. For this reason, some ERP vendors offer competing "ERP" products that focus on a particular subsystem/module. For example, Salesforce is a company that offers a CRM system, which has recently become one of the top CRM systems globally. Another example is Workday, which primarily provides human resource management modules with some financial planning modules as well. By focusing on just one or a few modules in the larger suite of ERP modules, these and other emerging companies and products are developing best-in-class offerings, taking market share from the large ERP providers like SAP and Oracle.

Manufacturing firms also require specialized ERP modules to manage manufacturing. Although some organizations are able to use "out-of-the-box" vendor solutions to manage manufacturing operations, many organizations require specialized manufacturing systems developed internally. Sometimes, "out-of-the-box" vendor solutions don't fit the intricate manufacturing processes adopted by organizations. When vendor solutions don't work with manufacturing processes, companies develop their own manufacturing management systems with the help of the IT department and/or an external consultant. Retail institutions also require specialized modules, such as point of sales systems that allow clerks or customers themselves (i.e., self-checkout systems) to make purchase transactions. In summary, there are many common business operations that are supported by mature information technologies.

## Emerging Information Technology in Organizations

Most of the information systems described in the previous section are used to collect, store, and provide reports about the operations of businesses to inform decision making among managers or other users. These systems require substantial human intervention in the form of data entry and human intuition in the interpretation of the information stored within the systems. Under the existing paradigm of information systems usage, humans make most of the important decisions in businesses based on information provided by ERP systems. That is changing slowly and will continue to change.

As an example, watch the following video provided by Amazon about the automation that takes place within their fulfillment centers: <https://www.youtube.com/watch?v=8nKPC-WmLiU>. In the video, many advanced technologies were described to show how Amazon is revolutionizing their business operations. Amazon is certainly not alone in this effort. Human involvement is still very necessary, but that need changes more each year.

Some of the major technologies that are substantially altering business operations include AI and machine learning, the Internet of things, and robotics. You do not need to have an intricate understanding of robotics, AI and machine learning, and other technologies to creatively consider how these technologies can be used to improve business processes or to develop new customer facing IT products and services. However, you should have an understanding of how they work and their limitations. In the following sections, some advanced technologies are described in high-level terms. This knowledge should only act as a starting point to further your study of these topics.

## ***Artificial intelligence and machine learning***

**Artificial intelligence** (AI) is the pursuit of computer systems that possess narrow, general, or even super intelligence. Narrow intelligence is the ability of an AI system to perform a single or small set of related tasks that are normally performed by humans (e.g., recognizing objects in an image, speech recognition, or speech translation). The AI currently in existence are forms of narrow intelligence. General intelligence refers to the ability of an AI system to respond with human level intelligence to a broad spectrum of tasks, including the ability to quickly learn to accomplish new tasks. Super intelligence is a step above general intelligence, granting the AI system superior intelligence to humans. Although some AI models can perform at superhuman ability, they only possess narrow intelligence. General and super intelligence have yet to be achieved in AI systems. **Machine learning** is a sub-field of AI that focuses on training computers to learn with data in a way similar to humans.

The goal of AI and machine learning experts is to produce computing programs that can make decisions without specific logic written by traditional software developers. Traditional software programs execute commands written by software developers in a linear fashion, exactly as the logic is written. Although there are many aspects of business that can be automated with simple linear instructions, information work (e.g., accounting, finance, marketing, etc.) is complex and requires dynamic and adaptive decision making. Some attempts have been made in the past to encode expert knowledge into traditional systems with hard coded, linear logic. These systems are referred to as **expert systems**.

Expert systems have traditionally been developed by working closely with domain experts (e.g., doctors, accountants, and financial planners) to identify the specific decisions these experts make each day. To do this well, experts must be interviewed to extract all of the implicit decisions they make to perform their work. Although this may sound simple in theory, in practice it is quite difficult. First, the way experts think may not even be consciously evident to the experts themselves. The human brain is still an enigma. Extracting the thought processes of domain experts is a difficult activity. Further, individuals are able to explain their decision processes best as they are performing the actual tasks. This requires IS professionals to interview experts as they work, which can sometimes be intrusive, such as in medical exams. It can also be dangerous if the expert requires extreme focus to complete their work. These and other difficulties have limited the ability to encode decision logic into traditional programs. Although it can be possible to provide simple intelligence through expert systems that encode decision making into

linearly coded logic, expert systems have not provided the advancements and support that many have hoped they would.

Other attempts to include “intelligence” in systems have included the use of data mining. **Data mining** is the use of statistical techniques to identify features and relationships within data that can be used to identify patterns that provide insight to business decisions about specific entities (e.g., specific customers) or entity groups (e.g., groups of related customers). In data mining, features are attributes within data, such as height, weight, age, income, etc. that predict some outcome variable, such as purchase behavior. Data mining relies heavily on human intervention (i.e., statisticians) to select the appropriate statistical techniques, to identify and extract features from data, and to identify patterns in data.

Once a set of features are selected, these features can be fed into custom designed statistical or algebraic models to identify patterns. The results of the models can be used to help organizations make decisions about specific customers, employees, business partners, etc. Data mining efforts often support human decision making, though they can be used for automation as well. Although data mining offers valuable insight to business operations and can be an important part of an organization's automation efforts, data mining isn't the artificial “intelligence” that many seek. With that said, familiarizing yourself with data mining will make you a valuable asset in leading automation. It is a topic worth understanding.

Machine learning seeks to move beyond traditional programming logic and data mining to give computing programs the ability to adapt to new situations and learn from successes and mistakes. It also seeks to bypass the need to manually extract features from datasets, as is required in data mining. At a high-level, many approaches have been developed to create “intelligent” systems.

### ***Applications of AI and machine learning***

Machine learning relies on a variety of learning approaches, such as supervised and unsupervised learning or other advanced approaches like self-supervised learning. **Supervised learning** requires a dataset consisting of raw input data and labeled outcome variables. The supervised learning algorithm uses the raw data and the labels to teach the system how to learn patterns in the raw input that relate to specific outcome labels. For example, if you wanted to develop a supervised learning algorithm to help computers identify and label real-world objects within images, you would need to provide images (i.e., the raw input data) and a label for the object(s) within each image (i.e., the output labels). An image of a cat would be accompanied with the label “cat” to teach the system to identify a cat from the pixels in the images.

Often, learning occurs within such algorithms by **adjusting numeric weights** within a statistical model until the weights reasonably reflect a connection between the input data and the correct label. Some complex supervised learning algorithms require thousands or millions of training data points to effectively adjust weights and accurately connect input data to the correct labels.

Supervised learning algorithms are used in a variety of business applications. A great deal of **computer vision** applications, which are systems with the ability to identify objects with a camera, rely on supervised learning, though many are also moving to self-supervised learning approaches. For example, supervised learning algorithms can be used to identify defects in products as they are manufactured. Supervised learning algorithms can be trained with pictures of defective and high-quality products with associated labels (i.e., “defective” or “quality”) until the model recognizes the difference between defective and quality products.

Once a model has been trained with a dataset of raw data and labels, it can be used by placing cameras within the manufacturing system to take pictures of products as they move through the manufacturing system. These images are then fed to the supervised learning model, which predicts whether the product is defective or not. Based on predictions of the model, the product would then continue through the manufacturing system or be diverted to a defect bin. Similar models are used in automated vehicles to help the vehicles stay within their lanes, parallel park, identify and avoid pedestrians, vehicles, and other obstacles, etc. Medical imaging also relies on supervised learning models to diagnose illness.

Supervised learning algorithms are also partially responsible for advances in applications that can understand human language. **Natural language processing** (NLP) is a large field of study within data mining and machine learning that seeks to understand and even produce human language in both written and oral forms. Chat bots are an example of such technologies. Chat bots are used by organizations to automate simple customer service interactions. Although some bots rely on traditional pre-programmed logic or hand designed data mining techniques, other bots, such as large language models, use NLP machine learning techniques.

Bots can respond to simple questions from customers and elevate concerns to humans if they are unable to provide appropriate answers. Advances in machine learning are leading to models of language that have allowed companies like Google to provide real-time language translation services and personal assistants. One day, you will place orders at fast food restaurants by talking to a bot. Many business transactions will proceed in this fashion. In fact, many customer service jobs may be automatable in the not so distant future with further advancement in NLP algorithms.

Supervised learning algorithms are also emerging in a variety of other industries and for various business purposes. The finance industry, using both data mining and machine learning techniques, develop supervised algorithms to identify good investment opportunities and to even automate the purchase of investments. Manufacturing floors use trained or mined models to identify when a manufacturing machine should be repaired to avoid mechanical failure by monitoring vibrations, heat signatures, the passage of time between repairs, etc. Human resource professionals are beginning to use machine learning systems that screen resumes and pre-recorded interviews submitted by applicants to identify individuals that could be a good fit for the company. The creative use of supervised learning algorithms are leading to a number of changes in the way businesses operate.

**Unsupervised learning** is another approach to machine learning that does not require labeled training data to help a system learn. Instead, the algorithm is fed raw data without labels and learns from patterns in the raw data. Collecting training data and then manually labeling data points for supervised learning is tedious and time consuming. It takes both time and money to train supervised learning algorithms. A system that can teach itself from raw data without labels reduces costs substantially. However, unsupervised learning algorithms are difficult to design and are often used for a limited set of problems. That is slowly changing.

Currently, unsupervised learning algorithms are primarily used for segmenting data. For example, organizations deal with a diverse set of customers. Some groups of customers have different preferences and expectations than other groups of customers. If an organization can efficiently address the customized preferences of different groups or individuals, the organization can better meet the needs of its customer base. If automated, it can do so at minimal cost. The first step in this process can be segmenting users to identify groups with similarities. Unsupervised learning can segment customers

based on data patterns. Although the future promise of unsupervised learning is something to look forward to, unsupervised learning algorithms require substantial advancement before we have systems that can learn a broad set of tasks without labeled datasets. One area where this is less true is NLP. Many advanced NLP techniques use self-supervised learning, a middle ground between supervised and unsupervised learning, to learn word patterns without the need for substantial data labeling.

Despite the difficulties that exist in advancing unsupervised learning algorithms, scientists have identified other methods to minimize the amount of supervised training data required to train a machine learning algorithm. Semi-supervised learning algorithms take advantage of strengths of both supervised and unsupervised algorithms. Semi-supervised learning algorithms require some labeled data, but not as much as is required for fully supervised machine learning models. Some of the semi-supervised learning models use labeled data in conjunction with an unsupervised learning algorithm to create labels for the unlabeled data in the dataset.

Scientists have also developed a method called **transfer learning**. Unfortunately, supervised learning models are only good at predicting outcomes for inputs that are reasonably similar to the training data. This means that an algorithm that was trained on images of defective and quality mugs might not be good at detecting defects in a similar type of product. However, there is still information contained within the model trained on mugs that could be used to lessen the amount of training data needed to detect defects in other products. A smaller set of labeled defective product images of another product category (e.g., cups or bowls) could be trained on top of the parts of the model that was trained to detect mug defects. In this way, the training data from one context can be “transferred” to minimize the amount of training required for another context.

The designations of supervised, unsupervised, self-supervised, and semi-supervised algorithms are only high level concepts. Models vary in their level of being supervised or unsupervised. Pay attention to advances in machine learning and artificial intelligence models. They will change the way you work in the not too distant future. Let's examine next how some of these models work.

### **Artificial neural networks and deep learning**

Artificial neural networks and deep learning are branches of machine learning that have received increased attention and implementation in recent years. These are not new concepts though. Work on deep learning stems back decades. It has only become a viable solution in recent years because of advances in the speed and storage capacity of modern computers. Deep learning is often supervised learning in practice, except for some natural language processing models that use self-supervised learning. So, neural networks and deep learning often require a substantial amount of labeled data (sometimes millions or billions of data points). Deep learning models also consist of many parameters and weights that have to be adjusted to train a model, sometimes millions or even billions of parameters/weights. This training is computationally expensive. Modern computing hardware can handle these intense training requirements. Your personal computer, on the other hand, might take days, weeks, months, or even years to train some deep learning models. The largest of models cannot even be trained on personal computers because they require so much memory and storage space. Smaller learning models can be trained on a personal computer.

Deep learning algorithms are called “deep” because they consist of many layers of computations. Deep learning algorithms were modeled after ideas from neurology regarding the function of the human brain (though deep learning is not a direct representation of the brain). Deep learning models often consist of

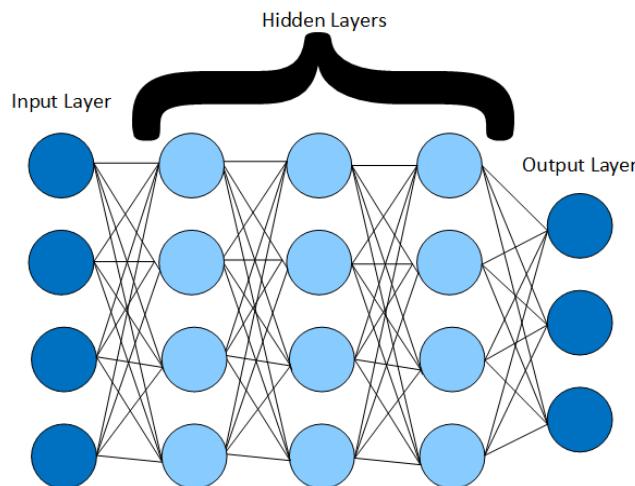
many layers of “neurons.” A **neuron** is simply a mathematical function that accepts an input value, applies a weight and bias value to the input value. For example, if the input value was 6 and the weight for the neuron was 0.52, the function might perform  $6 * 0.52$  to return 3.12. The weighted value would then be passed to an activation function. An **activation function** is often a nonlinear mathematical function, such as the sigmoid function or rectified linear unit (ReLU) function, that standardizes the weighted value between a certain range (e.g., between 0 and 1 or between -1 and 1). The resulting value of the activation function is then passed to a neuron or neurons in a deeper layer of the model and the process repeats until the last layer of neurons is reached.

This explanation was an oversimplification. There is more to a deep learning algorithm than this, and other types of deep learning models exist that don’t follow this model. But this explanation gives you a starting point for understanding the architecture of some common deep learning algorithms.

There are three basic types of layers in many deep learning models, each layer having many neurons. The **input layer** consists of neurons that receive the raw input data from a data set. The input layer neurons perform their weighting and activation functions on the data and then pass the output of the layer to neurons in the first **hidden layer**. Deep learning algorithms may have hundreds of hidden layers, each consisting of many neurons. The neurons in these hidden layers accept the data from neurons in the previous layer, run their weighting and activation functions on the data and pass the output to the next hidden layer. Finally, the last hidden layer passes its output to neurons in an **output layer**. The neurons in an output layer then make a prediction, such as whether an image shows a defect or not, or whether a pedestrian is in front of an automated vehicle or not.

The weights of each neuron in each layer of the model are often randomly assigned when the model starts its training. As the model receives more data, the weights in each neuron are adjusted based on the training data using other mathematical functions, such as gradient descent algorithms derived from calculus. It is this weight adjustment phase of the training that is computationally expensive, but ultimately leads to the model learning how to connect raw input data with the correct output labels. Figure 1.8 depicts a generalized representation of a neural network used for deep learning.

**Figure 1.8: A Generalized Representation of Deep Neural Networks**



How and what do neurons in these deep networks learn if all they do is perform a series of weighted calculations that are passed through an activation function? Testing of deep learning models shows that hidden layer neurons closest to the input layer learn simple features from the input data. For example, in an image, some of the first hidden layers are going to learn horizontal, vertical, and diagonal lines. Slightly deeper layers build on the simple learning of previous layers. For example, in an image, slightly deeper layers might learn curves and simple shapes. As the model progresses through deeper and deeper layers, the neurons in the layers learn more complex features. For example, in an image of faces, deeper layers might learn what a nose looks like or what an eye looks like. As the model progresses closer to the output layer, the neurons will learn about more complex concepts. In the case of images of faces, the deepest hidden layers might learn to recognize an entire face. It is amazing what can be learned by repeating simple calculations multiple times across interconnected layers of neurons utilizing nonlinear activation functions.

Implementations of neural networks and deep learning are diverse. They come in many forms. The description above is a generalization of deep learning architectures. Some different deep learning architectures that are proving useful for business applications include **convolutional neural networks** (CNNs), **recurrent neural networks** (RNNs), **generative adversarial networks** (GAN), and transformers. Importantly, each of these categories of deep learning can be implemented in a variety of ways. Research into the creation and use of different types of CNNs, RNNs, GANs, transformers and other deep learning architectures continues at a rapid pace. Although each of these architectures can be configured in many ways, each architectural type represents different ways of approaching machine learning.

CNNs have proven particularly useful for computer vision problems. CNNs scan the pixels in an image in small windowed steps and apply a filter (i.e., a weighting schema) at each step. These filtered values are passed to deeper layers of the CNN model where they are compressed. These basic functions are repeated multiple times over many layers. As described above, early layers learn simple lines, then simple shapes, more complex shapes, and finally entire objects. RNNs and transformers have proven useful for time series data (i.e., data that occurs over time). For example, language models often require RNNs or transformers, because the sequence of words in a sentence matters. RNNs differ from CNNs in that RNNs possess a form of “memory” in which the layers of the model keep track of what values were learned in previous layers. GANs are used to generate new outputs after being trained on a set of inputs. CNNs and RNNs are primarily used for classification and prediction problems.

GANs and some transformers are used to generate new data. For example, GANs and transforms have been developed to write simple stories, paint pictures, generate audio output, and more. For example, the company OpenAI has created a transformer (i.e., Dalle-2) based model that can create novel images based on users’ textual prompts. You can view Dalle-2 at: <https://openai.com/dalle-2/>. Dalle-2 and future iterations of similar technology will revolutionize the marketing industry, the movie industry, and more. You may one day be able to develop your own feature length film from the comfort of your own home. Similarly, large language models like ChatGPT rely on transformers to generate text in response to some input.

When used together in a system, these various types of deep learning models can produce interesting behavior. Some complex systems, like AlphaGo that beat a human player at the game of Go, are made up of several different types of deep learning models that work in conjunction to produce a certain behavior. Similarly, vehicle automation is managed by many different subsystems made up of a variety of machine learning techniques mixed with traditional flow control programming logic. ChatGPT and other

large language models also consist of multiple different types of models, such as transformers to learn the basic structure of language and relationships between concepts, and reinforcement learning to continually improve the outcomes of what is generated by the models.

There are many articles and resources online to help you learn about these models and remain alerted to the latest advancements in deep learning and neural architectures. You don't necessarily need to understand the intricacies of these models. Rather, you need to understand the potentials and limitations of the models. Currently, many of these models possess serious issues that you should be aware of as you try to apply them to business applications. For example, simple "***noise*** can drastically alter the outputs of a machine learning model. For example, images that look like static on old televisions (i.e., noise) can be overlaid on an image to create incorrect predictions. For example, a CNN can be made to think that a cat is a pig if the right noise is applied to an image of a cat. The difference produced by the noise wouldn't be perceptible to the human eye, but it can alter machine learning predictions substantially. This means that deep learning algorithms can be hacked to produce unexpected and even dangerous results. By adding the correct noise to an image of a check, a system could be made to read \$10,000 instead of \$1,000. This security vulnerability requires caution in the use of deep learning models for business applications. Developers and security experts will need to carefully plan for the use of artificial intelligence in integral financial systems.

As mentioned previously, many of these models also fail to handle novel situations they were not trained to handle. Some of the crashes that were caused by automated vehicles occurred because the learning models were not trained on all possible road conditions or because road conditions changed (e.g., new construction). The vehicles were not sure how to react to the new road conditions, causing them to behave erratically.

Further, deep learning models can learn biased perceptions through the training data they are fed. For example, a model fed images of men with few women or a model fed data that doesn't contain racial diversity may produce biased predictions when put into practice. Some early automated hiring models in industry have been shown to possess bias that deprioritize women and minority groups. Great care must be taken when training deep learning models. A machine learning model is only as good and unbiased as the data you train it on. As a business professional, you should be a voice of reason to help technical experts choose appropriate training data. You must think critically about what implicit biases lie within your business datasets. This requires domain expertise and an ethical and critical eye.

## ***Reinforcement learning***

Another machine learning model that deserves attention is reinforcement learning. **Reinforcement learning** is a machine learning model in which an automated agent learns by exploring an environment by enacting specific actions. As the agent explores the environment and takes actions, some actions lead to rewards and others do not. Over time, the agent learns a policy to describe what actions lead to rewards and/or prevent penalties and prioritizes those actions.

Currently, reinforcement learning requires the automated agent to engage in a great deal of experimentation. Sometimes, the actor must engage in millions of actions before learning even simple behaviors. To date, reinforcement learning has primarily been used to automate the game play of video games. The maps and levels of the video game represent the "environment," the character in the video game represents the "actor," and the reward or punishment might be related to elements of the game play, such as collecting coins, the character being injured, etc. These video game applications have been

used as a proof of concept to show the potentials of reinforcement learning. However, reinforcement learning is proving useful for large language models to ensure that the content generated from the models is acceptable to users. Through reinforcement learning, the outputs of large language models can be improved to meet the expectations of those using the models.

Despite some of these advances, reinforcement learning hasn't made as big of an impact on practice as supervised and self-supervised learning yet, simply because of the amount of trial and error required to train a reinforcement learning algorithm. In the real world, managers don't want to make poor investment decisions through trial and error. Similarly, most companies don't have millions of trials to learn from. Last, many reinforcement learning algorithms have difficulties learning when a reward received from taking an action is delayed. In the real world, many rewards come after significant time lags. These limitations have kept reinforcement learning in the shadows of practice. However, there are many researchers trying to break past these various limitations. For example, there are some algorithms that are better at learning when rewards for actions are delayed. Similarly, some have overcome the limitations of needing multiple trials by creating two autonomous agents and letting them "play" against each other. In fact, this is in part how AlphaGo was created. Reinforcement learning can be tied to other types of machine learning, such as supervised and unsupervised learning.

Much still needs to be done to advance reinforcement learning algorithms to meet the needs of industry. However, it does provide interesting avenues for practice. For example, a reinforcement learning algorithm could one day be used to learn stock trading techniques, in which the automated agent takes buy/sell/hold actions within the market environment, monitoring its rewards (i.e., gains and losses). Again, one must be careful when releasing an untrained reinforcement learning algorithm into the "wilds" of industry. It could be easy to lose a fortune using a machine learning algorithm that learns through trial and error. It is possible to train these models on past data or replicated data, but these approaches don't always map to the data found in practice. Machine learning in its various forms promises to provide many advances in the form of automation. However, these advances need to be approached thoughtfully.

## ***The Internet of Things***

The Internet of things (IoT) promises to provide fine-scale data about business operations. IoT refers to the use of sensors with Internet connectivity that can communicate with one another or with centralized systems that make decisions based upon the information provided by the sensors. Humans possess many senses that allow us to interact with the world in intelligent ways. It is in part through sensing that we are able to learn. It is the same for machine learning algorithms. These algorithms require data about the environment in which they operate to be trained to make good decisions once deployed in practice.

Sensors are crucial to many emerging technologies, for example smart vehicles. There are many types of sensors that provide detailed information about the environment. For example, a car requires visual senses. Although this could be accomplished through standard cameras, cameras can become impaired by dust, water, and other environmental conditions, and may not function well at night. Thus, many smart cars use lidar and other vision sensing technologies to measure distance between objects and to see the world from a variety of perspectives. Combined, these various sensors provide intricate data about the environment. Other sensors determine position in space (i.e., upside down or right side up), sound, temperature, humidity, movement, and even smell. These various sensors can be used by creative designers to give products sensing abilities beyond even what a human possesses. Some humans are even using sensing technologies to augment their own senses.

IoT provides the sensing ability that machines require to learn about their environment and make automated decisions. Internet connected sensors are used in a variety of industries to collect data that are used to train data mining or machine learning models that are then put into practice, utilizing the same sensors to make automated decisions. Such applications are becoming more common in a variety of industries.

As mentioned above, the automotive industry is using sensing technology and machine learning to help cars make decisions about how to drive. At the moment, many of these decisions are simple, such as parallel parking, lane assist technologies that keep you within your lane, etc. However, the industry is pushing toward fully automated vehicles. Sensors play an extremely important role in making these mobility technologies safe, reliable, and capable of handling a failure in a single sensor. Changes in the automotive industry will have broad effects on how we drive, whether we even need to own a car, real estate requirements for parking, and the transportation of products.

Similarly, sensors and machine learning are being used in industry to help with the transfer of goods from manufacturer to consumer. Supply chains are complex and require the effort of many organizations working in synchronization to produce the right amount of goods and move those goods to the appropriate locations. Sensors in trucks, such as GPS sensors, can help track shipments. Further, radio frequency identification (RFID) chips and similar technology can be used to track independent parts and products. Vision and weight sensors can help to determine whether retailers need to order more inventory. As a future manager, you may lead the way at some point in integrating systems with sensors that allow for even more nuanced distribution of goods.

Sensors are also used by governments and companies for critical infrastructure. Developing smart roadways and electrical grids, for example, is a move toward smart cities. Traffic lights, for example, can be equipped with various sensors to make real time decisions about when traffic lights should change to optimize traffic flow. Electrical grids can minimize the loss of power by carefully monitoring the demand for electricity in certain parts of a city through the use of specialized sensors spread across the grid and machine learning algorithms. Many other aspects of government will soon rely on sensors and machine learning to automate decisions to manage various aspects of modern city life. Caution should be taken here though as sensors can allow for the tracking of individuals, not unlike the dystopian society described in the book 1984.

One last important industry that has received important advances in sensing technology includes the medical industry. Medical technologies, such as pacemakers, are implanted in the body to monitor health conditions and to even help individuals maintain healthy bodily functions. These devices can be connected to the Internet to provide real-time analytics to doctors and healthcare systems. Such advances in the medical field are allowing changes in health to be caught early to provide important interventions before conditions worsen. Other technologies even provide real-time correction of medical conditions by sensing changes in organ function and then providing automated treatment. Some such technologies are experimental. The medical field continues to push the boundaries of what can be sensed within the human body, and what health decisions can and should be automated.

### ***Ethical considerations of sensing technologies***

Sensing technologies have come a long way and are becoming integrated into every part of life. With new technology often comes new ethical dilemmas. For example, sensing technologies allow governments and organizations to track the intricate movements and decisions of citizens, employees, and customers.

Although this tracking can lead to better service, it can also be viewed as an invasion of privacy or a means of manipulation.

Similarly, information is crucial to the development of power and control. Imbalances of information can allow one group to leverage control over another. With large amounts of information being consolidated into the hands of a few organizations and governments, the potential for the abuse of power increases, unless the same technologies are also used to track and report on these governing bodies.

It is also possible to defraud these automated sensing technologies if proper redundancies are not put into place. For example, it could be possible to ship an empty crate of RFID chips not attached to parts or products. Thus, it would appear to an automated system that the shipment had arrived. However, the actual products would not be found. Sensing technologies often require auditing from other sensors and/or humans to provide accurate information and to prevent fraudulent behavior. In another example, lidar technology provides an audit to simple camera inputs to provide a more detailed and correct visual perception of an environment. Without such redundancies, human life can be lost and businesses and consumers can be defrauded.

Concern also exists about the potential for hacking. Internet connected sensors provide hackers with additional computing devices to manipulate. Given that sensors help to make automated decisions, the potential for damage is large. For example, an Internet connected health device embedded in the human body could be hacked if not properly designed. Hacking devices embedded within humans could lead to health conditions or even death. Similarly, hacking a smart vehicle could allow hackers to engage in nefarious acts, such as redirecting vehicles or even causing accidents intentionally. On a less damaging scale, sensing devices like Google Home, Alexa, and other smart assistant technologies could be used by companies, hackers, or governments to monitor individuals without consent. The potential for harm is great. As future managers and leaders of society, you will be called upon to address these and other ethical dilemmas.

## **Robotics**

Advances in robotics represent other emerging technologies that with machine learning and IoT are making major changes to the way businesses operate. Intellectual work will primarily be automated through the use of machine learning and IoT. Physical work, however, requires a physical agent to perform physical actions. Robotics fills this role. Robotics has traditionally been an engineering discipline, with programming existing in the form of coded scripts to control motors. With advances in machine learning and different sensing technologies, programs that control motors are increasingly complex.

For example, with advances in computer vision, robots have an increasingly complex visual sense of the world. In the video about Amazon's fulfillment centers shared earlier, robots are used to move shelves of products throughout the warehouse. These robots need a sense of where they are and where they are going. This sense is provided in part by cameras and computer vision machine learning algorithms that allow the robots to navigate the warehouse floor autonomously. Similarly, embedding cameras and machine learning algorithms on manufacturing lines can allow for decisions to be made autonomously as products move down the line. As mentioned earlier, computer vision can be used to identify defects and divert defective products by way of automated mechanical devices.

Similarly, advances in robotic movement are clearing the way for increasingly agile robots. For example, Boston Dynamics has built Atlas, Spot, and other robots with advanced movement capabilities. Although

it may seem simple to make a robot move or walk, balance has been problematic for many decades. When a robot requires dexterity or is required to move heavy objects, it needs balance. To get a feel for some of the emerging capabilities of agile robot movements, see this video from Boston Dynamics of various robotic designs dancing to “Do you Love Me?”:

<https://www.youtube.com/watch?v=fn3KWM1kuAw>.

With advances in movement, tied to advances in computer vision, robots will be increasingly able to make automated decisions in the physical world. In the future, robots like these may be able to perform dangerous tasks with human-like decision capabilities that will minimize death and injury inherent in certain types of work. However, this level of automation is years to decades away.

Robots are also being used to support mental health. Stress, anxiety, and loneliness are increasingly common in our modern world. Many researchers and companies are developing intelligent robots that can react to the mental health needs of children, employees, and the elderly amongst other groups. Some are exploring the design of machine learning algorithms to detect mood. Others are exploring machine learning algorithms that can provide physical or verbal responses to support individuals experiencing anxiety and depression. Although some of this can be done without robotics, the presence of a physical robotic element may make these efforts more meaningful. In some cases, the robotic element is designed as a support animal. Beyond making decisions, managers are often called upon to emotionally support employees. In the future, this emotional support may be provided by robots.

Robotics is advancing in ways, which in the coming decades, will result in the automation of many types of physical labor. Although this is in part due to advances in robotics and control systems, machine learning and Internet connected sensing technologies will allow for robots in various forms to complete many physical tasks that were previously not automatable. Taking part in the design of the future of work will ensure that you stay ahead of the rapid changes taking place in industry.

## ***Other Emerging Technologies***

Although we have highlighted AI and machine learning, IoT, and robotics, many other emerging technologies will lead to substantive changes in the way we work and live.

For example, blockchain technologies provide a ledger system that keeps track of business transactions through a decentralized network of computers. You are probably most familiar with blockchain technology in the form of cryptocurrencies like Bitcoin. Blockchains store information in “blocks” of data. When a block of data is full, the block is attached to a previous block that forms a chain of transaction blocks. When information is added to a block, it is timestamped, meaning that the chain of blocks represents a chronological record of all of the transactions stored within the blockchain. Traditional databases are often stored on centralized servers (i.e., powerful computers) owned or leased by a company. Conversely, blockchains are often built on a decentralized architecture where many companies or individuals may own the servers/devices that host the data. By decentralizing the database (i.e., ledger), a company cannot fraudulently change its records because the ledger of transactions is shared by many companies or individuals. The decentralized nature of blockchains increases the certainty that a transaction truly takes place, as each transaction is verified by many distinct actors.

Quantum computers are another emerging technology that may fundamentally change the speed at which technological advancement takes place. Quantum computers take advantage of the properties of photons and other quantum elements to store and compute information. Traditional computers rely on bits of 1 or

0 (on or off) by way of electrical pulses. These electrical pulses are either translated to a 1 or 0. With quantum computers, photons can be in similar positions 1 or 0, or in a superposition of 1 and 0 simultaneously. Instead of bits, quantum computers operate on qubits. This ability, along with other properties of photons provide the potential for quantum computers to make calculations more quickly than traditional computers. They also open the exploration of quantum concepts that traditional computers are not able to process.

Unfortunately, quantum computers often require specialized environments to function (i.e., extremely cold temperatures) and imprecise outcomes that must be computed several times to reach a desired level of precision. Quantum computers also have fewer qubits to operate on than traditional computers have bits. As such, there are currently few tasks that quantum computers can perform faster than traditional computers. However, advances are identified regularly which increase the number of qubits within quantum computers and enhance the precision of calculations. With time, quantum computers may lead to major advances in many scientific fields, particularly those that rely on quantum physics.

With the many technologies emerging in industry, you must be prepared for change. You must pay attention to advances in technology within your field and related fields. Failure to carefully consider the changes that will take place in industry over the next several decades could lead to many failing companies and you in an obsolete career path. Provide yourself with time to read about advances in technology to prepare yourself for changes to come. Examine the properties of emerging technologies and consider how they might affect your field. Doing so will help you to manage the waves of automation that will take root across the globe over the next several decades.

## Chapter 1 Assignment

Please see the [Chapter Assignments Workbook](#) for the Chapter assignment.

# Chapter 2:

## Programming Fundamentals

Now that you possess an understanding about what information systems are and of existing and emerging information technologies used for business, we can begin to explore foundational programming concepts. In this chapter, you will learn some basic Python programming syntax to write simple programs using procedural and object-oriented programming techniques.

Many programming paradigms exist to write software. A **programming paradigm** is a way to think about writing computer programs to solve problems. Python is a general purpose programming language that provides at least some support for multiple programming paradigms, including functional programming, procedural programming, and object-oriented programming. In this text, we will primarily explore procedural programming and object-oriented programming.

One common feature found among functional, procedural, and object-oriented programming paradigms is the use of variables. Given their common importance, we will begin our discussion of programming with the topic of variables.

### Variables in Programming

You have probably heard the word “variable” before and you may have different understandings of what the term means. For example, if you have taken an Algebra class, you learned mathematical equations and functions that rely on variables to represent unknown values that can vary. In Algebra, these variables tend to be immutable. An **immutable variable** is one whose value does not change after it is initially set. In other words, if a variable *x* is set to 6, then *x* will retain the value 6 through the entire execution of the mathematical function. In programming languages like Python, it is possible to create immutable variables, but they don’t always behave exactly as an immutable variable in Algebra. Immutable variables are important when performing mathematical operations with a computer program. However, you will find that most variables in computer programs like Python tend to behave more like mutable variables in practice. A **mutable variable** is one whose value can be altered during its usage.

In statistics, you may have heard the term “variable” used in relation to predictor and outcome variables. A **predictor variable** (i.e., independent variable) is a data feature like age, height, gender, or weight that could explain or predict the value of an **outcome variable** (i.e., dependent variable) like salary. In business analytics and machine learning, predictor variables are sometimes called **features** and outcome variables are sometimes referred to as **labels**. In later chapters, we will explore variables from the perspective of statistics and machine learning. However, this is not the definition we will use when discussing programming concepts.

In terms of Python programming, we will primarily think of “**variables** as named storage buckets in **computer memory**”. Like your own short-term memory, computers have short-term memory called random access memory (RAM). RAM is used to store and retrieve data so that computer programs can perform calculations on data efficiently. Variables that you create in a computer program reserve space in memory to store data values.

In many algebraic equations, variables are named with simple letters (e.g., x, y, z) or Greek symbols (e.g.,  $\Sigma$ ,  $\sigma$ ,  $\pi$ ). Although you will see simple letters used from time to time in computer programs, more often, you will see **variables named for human readability**. For example, if you want to store the age of a customer in a variable (i.e., a named storage bucket in memory), you would likely name the variable “age.” Similarly, a variable used to store a customer’s first name might be named firstName or first\_name. In each case, the variable you create has a name that references a space in memory where the data is stored. When you want to retrieve data from a variable, you simply call the name of the variable and the computer program will retrieve the value stored in the associated memory location.

When **naming variables**, don’t use spaces (at least for Python and many other languages). Software developers use different conventions for naming variables. You will see some developers use **underscores** where a space would exist. For example, “first name” would be named first\_name. Other developers will use **camel case** to differentiate multiple words in a variable name. In camel case, the first word is lowercase. Then, the first letter of every subsequent word is capitalized with no spaces between words. For example, “first name” would be named firstName. Adding spaces to variable names in Python will result in errors in your code. Be cautious when naming variables.

## Variables and data types

When you create a variable to store data, you usually know what type of data you want to store in the variable. For example, a variable named “age” would likely be used to store an integer representing a user’s current age. A variable named “height” might be used to store a decimal value representing a user’s height in inches or centimeters. A variable named “name” might contain a string (i.e., textual data) containing the user’s first and last name. In each of these examples, a different type of data was stored in the variable.

There are several data types that together are often referred to as primitive data types. **Primitive data types** store single data values. In Python, primitive data types include:

- **Integer** - positive and negative whole numbers (e.g., -1, 0, 1, 5000, -30000). As depicted in some of the examples, large integers do not contain commas in programming languages as they often do when we write integer values in documents.
- **Float** - floating point numbers, like decimals (e.g., 1.25, 5000.1, -45.3). Again, large floating point numbers do not contain commas in many programming languages.
- **Boolean** - true or false values
- **String** - textual characters (e.g., numbers, letters, punctuation marks, etc.). String values are contained between quotes in many programming languages (e.g., “hello world”, “my name is Jeff”, etc.).

Python is a **dynamically typed** programming language, which means you don’t have to specify what type of data a variable will hold. Any variable you create in Python can hold any data type. You can also change the data type of a variable in Python as the program executes (though this is dangerous and bad practice). The Python interpreter, which runs your code, makes an educated guess about the type of data that is stored in a variable. For example, if you had a variable named “age” and put the value 21 in the variable, the Python interpreter would likely type the variable as an integer. If you put the value 21.5 in the same variable, the interpreter would likely type the variable as a float. If you put the value “twenty-five” in the variable, it would likely be typed as a string. Although you don’t specify a data type for variables in Python, you should carefully consider the type of data that should be entered into each variable. Strange

errors can occur in programs when you fail to pay attention to the type of data stored in a variable. For example, you can't multiply the number 5 by the string "twenty-seven".

Importantly, not all programming languages are dynamically typed. Some languages, like Java, are **statically typed** programming languages. In these languages a specific data type must be identified for each variable you create. In statically typed languages, an error will occur if a variable that is typed as a specific data type receives a data value that is not of that type. For example, a variable that is typed as an integer and receives a decimal value would create an error in a statically typed language. The data type and the value stored in the variable must match in these languages. Statically typed languages provide protection against illegal values entering your program. For example, it isn't possible to multiply an integer by a string (i.e., textual data) in statically typed languages. However, in Python, it would be possible to make this mistake. Because Python doesn't have strong data typing protections, you could make the mistake of trying to multiply an integer by a string. **You must be careful when using variables in dynamically typed programming languages!**

When creating variables, developers use the term, "declare a variable." **Declaring a variable** means to reserve space in memory for data to be stored with the specified name of the variable. In statically typed languages like Java, you also set the data type of the variable when declaring it. Data type declarations are not required in many dynamically typed languages like Python. Variable declarations are a type of **statement**. A statement simply identifies an action to be performed by the program.

You will also hear the term "assignment" in relation to variables. **Variable assignment** refers to setting a value for a variable that was previously declared (i.e., `myName = "Jeff"`) or assigning a variable with a new value. In Python, you can assign a variable with different values throughout the steps executed by your program.

Often, you will want to declare a variable and assign it a value at the same time. This is known as **initializing a variable**. In Python, variable initialization is the norm. In fact, Python does not allow for true variable declarations. The closest equivalent to variable declaration in Python would be setting a variable to None (i.e., empty). For example, `myName = None`. Most variables in Python are created through initialization techniques (e.g., `myName = "Jeff"`).

The example below shows how variables are initialized in Python. You will notice that different data types are used for some of the variables (e.g., integers, floats, booleans, and strings). The final variable in the example, "BMI," stores the value of a calculation that relies on the values stored in the weight and height variables to store the calculated BMI of the user.

Numeric variables can be used with basic mathematical operators to perform business logic. The **basic mathematical operators** in Python are:

- Addition: + (e.g., six plus seven is: `6 + 7`)
- Subtraction: - (e.g., six minus seven is: `6 - 7`)
- Multiplication: \* (e.g., six times seven is: `6 * 7`)
- Division: / (e.g., six divided by seven is: `6 / 7`)
- Power of: \*\* (e.g., six raised to the power of seven is: `6 ** 7`)

## Examples of Variables in Python

```
#A variable named age that stores the integer value 21 in memory  
age = 21

#A variable named height that stores the float value 69.5 in memory  
height = 69.5

#A variable named weight that stores the float value 179.5 in memory  
weight = 179.6

#A variable named hasDriversLicense that stores a boolean value True in memory  
hasDriversLicense = True

#A variable named name that stores the string value "Jeff Wall" in memory  
name = "Jeff Wall"

#A variable named BMI that uses other variables to calculate the BMI of a user  
#Simple BMI calculation: (weight (lb) / height (in) / height (in)) x 703  
#The value stored in BMI would be a float calculated by: (179.5 / 69.5 / 69.5) * 703 = 26.139  
BMI = (weight / height / height) * 703
```

Notice that in the example above **variables** are displayed in **red text** and **variable values** are displayed in **green text**. This convention will be used throughout this text to help you recognize the different parts of the program. Some IDEs will also color code different parts of a program to help you read your code more easily. With an understanding of variables, procedural programming will now be introduced.

## Procedural Programming

**Procedural programming** is a paradigm that seeks to encapsulate business logic in blocks of code called functions, procedures, or subroutines. Procedural programming relies on **functions** to contain specific business logic that can be executed over and over again as needed. In procedural programming, the logic within a function is executed sequentially line by line. For example, a simple function could be written to calculate the value of a stock portfolio by multiplying the quantity of each company stock in the portfolio by the stock's price and then adding together the resulting values from each company stock. Each time you want to check the portfolio's value, you would simply call the function.

A procedurally programmed software system would consist of numerous functions that could be executed to perform user requests. For example, one function could be written to purchase a new stock for a portfolio. Another function could be written to sell a stock within a portfolio. In reality, each of these functions might need to be broken down into other smaller functions, such as connecting to an online broker, checking whether sufficient funds exist to make a purchase, etc. In procedural programming, multiple functions can be called in sequence to perform complex logic that a single function could not perform alone. Many business analytics professionals write their initial data analysis using procedural programming techniques, which is why it is introduced here.

## Writing functions: parameters, logic, and return values

**Functions** are like business processes in that they **accept inputs, process the inputs, and return outputs**. In programming lingo, the inputs of a function are called **parameters**. Parameters are simply variables used to store values that will be passed into the function. The output of a function is called the **return value**. In Python, the output of a function is returned to the user through a **return statement**. The return statement uses the keyword “return” followed by the variable or value that you wish to return. The process of changing the input variables (i.e. parameters) into an output (i.e., return value) is contained within the body of a function. The return statement also exists in the body of the function, typically at the end of the function body.

In Python, all functions start with the keyword “**def**,” followed by the name of the function, a comma separated list of parameters (i.e., input variables) contained within parentheses, and a colon to end the line. Together, these elements make up what is known as the **function signature** (a.k.a., method signature). For reference, the keyword “def” is short for “define” the function. In Python, the body of a function is where the logic resides. The **function body** starts on the line after the function signature and is **indented** with a tab on the keyboard. The body can contain multiple lines of logic, but each new line of logic must be indented.

### Syntax for the Signature of a Function in Python

```
def functionName(parameter1, parameter2,... parameterN):
    #body of the function is indented on the next lines.
    #The logic and the return statement for the function would go here
```

The code below shows a simple function called `addTwoNumbers()` that takes two numbers as parameters (i.e., inputs). The function body then adds the two numbers together and returns the value of the summation using a return statement. You will notice in the previous two examples that some lines start with a hashtag symbol (i.e., `#`). The **# symbol** is used to tell a Python program to ignore whatever comes after the `#` symbol on that line. The `#` symbol allows developers to write human readable comments about the code to explain what is happening to other developers who might need to use or change the code later.

### Example of Creating a Simple Function in Python

```
#This is a comment that is ignored when the software program is run
#The name of this function is addTwoNumbers(). Notice it starts with the keyword "def"
#The function accepts two parameters (i.e., inputs): firstNumber and secondNumber
def addTwoNumbers(firstNumber, secondNumber):
    #This is the body of the function where inputs are changed to outputs
    #The firstNumber is added to the secondNumber
```

```
#The result of the summation is stored in a variable called "result"
result = firstNumber + secondNumber

#the return statement below returns the value stored in the "result" variable
return result
```

Notice in the example above the **functions** are displayed in blue text. Also notice that red text is used again to represent the different variables used within the function. As depicted with the color coding in the example, function parameters are simply variables that store the input values passed into a function.

## Calling a function

If you were to run the code in the example above, nothing would happen. Functions are meant to encapsulate logic that can be used multiple times with different input values. Notice in the example above that the parameters do not contain specific values, just a variable name (i.e., a parameter variable declaration).

**Each time you wish to use a function to produce an output value, you must call the name of the function and pass it specific input values.** To call a function that you created, simply reference the name of the function followed by parentheses and values for any function parameters. If parameters exist, you may be required to pass the function an argument for each parameter the function expects to receive. An **argument** is a data value that is passed into a function parameter.

In the example below, the same summation function is created as in the previous example. Then, the function is called twice, passing in different arguments for the parameters each time it is called. In the first instance, the argument 4 is passed into the firstNumber parameter and the argument 8 is passed into the secondNumber parameter. In the second instance, the argument 6 is passed into the firstNumber parameter and the argument 10 is passed into the secondNumber parameter. In this way, the same function can be used multiple times to produce different outputs.

### Example of Using a Simple Function in Python

```
#As before, a function called addTwoNumbers() is created
def addTwoNumbers(firstNumber, secondNumber):

    result = firstNumber + secondNumber
    return result

#To call (i.e., use) the function and return a result, reference its name and pass it arguments
#The result of the function is stored in a variable called value1
value1 = addTwoNumbers(4, 8) #this function call would return 12

#The function can be called again with different arguments
#The result of the function is stored in a variable called value2
value2 = addTwoNumbers(6, 10) #this function call would return 16
```

## **Ordered vs. named parameters in function calls**

In the example above, the calls to addTwoNumbers() possess arguments in a specific order. For example, the 4 in the first call to addTwoNumbers() gets stored in the firstNumber parameter and 8 is stored in the secondNumber variable. The same is true for 6 and 10. When passing arguments into a function, you will use the order the parameters were written in the original function declaration. Using arguments and parameters in this way is called **ordered parameters**. Many programming languages, such as Java, only allow for the use of ordered parameters.

However, Python also allows for the use of named parameters. **Named parameters** look like variable assignments. When using the named parameter style, you can pass arguments to a function in any order by calling to the name of the parameter you want to pass a value to. You can even combine ordered and named parameters in a single function call with Python. In the example below, you will see the use of ordered parameters, named parameters, and a combination of ordered and named parameters.

### **Example of Using a Simple Function in Python with Ordered Parameters**

```
#A function called addThreeNumbers() is created
def addThreeNumbers(firstNumber, secondNumber, thirdNumber):

    result = firstNumber + secondNumber + thirdNumber
    return result

#The example below uses the function with only ordered parameters
value1 = addThreeNumbers(4, 8, 2) #this function call would return 14

#The example below uses the function with only named parameters in a different order
value2 = addThreeNumbers(secondNumber=10, firstNumber=6, thirdNumber=2) #this would return 18

#The example below uses the function with ordered and named parameters in a different order
#The value 12 would be put into the first parameter (i.e., firstNumber).
#The other values are assigned using the parameter names.
value3 = addThreeNumbers(12, thirdNumber=8, secondNumber=2) #this would return 22
```

In later chapters, you will see the value of using named parameters when you use functions created by others to perform analytics and machine learning calculations. For now, just recognize that there are two ways to enter arguments into a function call.

## **Using built-in Python functions**

Python has many functions that are built into the language itself, which means that you don't always need to create your own functions to perform a desired behavior. For example, you might want to display information to users after performing a calculation. Python has a built-in **print()** function that can be called to display information to the user. To use a built-in function, you simply need to know the name of the function and the arguments that it expects to receive. For example, the print() function expects to receive an argument to display to the user.

### Example of Using the Built-in print() Function in Python

```
def addTwoNumbers(firstNumber, secondNumber):  
    result = firstNumber + secondNumber  
    return result  
  
value1 = addTwoNumbers(4, 8)  
value2 = addTwoNumbers(6, 10)  
  
#Notice that built-in functions can be called multiple times like the custom functions you create  
print(value1) #this would display 12 to the user  
print(value2) #this would display 16 to the user
```

Notice in the example above that the results of one function (i.e., `addTwoNumbers()`) is passed to another function (i.e., `print()`). By combining functions in this way, programs can perform complex behavior.

Procedural programming can be a useful paradigm for creating analytics programs. Throughout this text, you will see procedural programming examples that increase in complexity. For now, recognize that a function accepts inputs, performs logic on those inputs and returns the outputs of that logic. Also recognize that a function must be called and passed required arguments to produce an output.

### Variable scope in procedural programs

When you create a variable in a program to store a value in memory, the variable can only be accessed within certain parts of the program. We call the limits of where a variable can be accessed within a program the **scope of the variable**. In procedural programming with Python, variables are used to represent the parameters of a function and to store values within functions and outside of functions.

A **parameter variable** can only be accessed from **within the body of the function** that owns the parameter (i.e., where the variable was declared or initialized). The parameter variable cannot be accessed outside of the function or from within other functions. A **variable created within a function** can also only be **used within the function** that owns the variable. It cannot be used within another function or outside of the function. In Python, **variables contained outside of a function** can be **used outside of the function or within any other function**. The examples below depict these scenarios.

### Example of Variable Scope

```
#variable1 can be used anywhere in this program  
variable1 = 4  
  
#parameterA can only be used within the body of function1()  
def function1(parameterA):  
  
    #The line below would be okay because parameterA can be used within function1()  
    print(parameterA)
```

```
#The line below would be okay because variable1 can be used anywhere in the program
print(variable1)

#result can only be used within the body of function1()
result = parameterA + variable1
return result

#parameterB can only be used within the body of function2()
def function2(parameterB):

    #The line below would throw an error because parameterA can only be used in function1()
    print(parameterA)

    #result is okay to use again here because both "result" variables are scoped to their functions
    #Even though okay, this is not always best practice. When it makes sense, use distinct names.
    #Using variable1 below is okay because variable1 can be used anywhere.
    result = parameterB + variable1
    return result

#This function call would print correctly because all of the variables in the function are scoped correctly
print(function1(6))

#This function call would throw an error because parameterA is not scoped correctly within function2()
print(function2(8))

#The line below would print correctly because variable1 can be used anywhere in the program
print(variable1)

#The line below would throw an error because parameterA is cannot be used outside of function1()
print(parameterA)

#The line below would throw an error because result cannot be used outside of the respective functions
print(result)
```

## Using variables within strings

You will often want to display the results of your programming calculations with users. This can be done with visualizations, such as graphs and charts. Displaying results is also done through text (i.e., strings). Although it is possible to simply display a number to a user (e.g., 24), numbers by themselves lack context. By wrapping calculated numbers within a sentence, you can contextualize your results to improve interpretability. This can be done using the knowledge you have of variables, data types, and functions.

For example, if you were writing a script to calculate the value of a portfolio of stocks, you could return some numerical value, such as 23654.84. This approach requires that those reading the results know what 23654.84 refers to. Is the value the initial investment value of the portfolio, the current value of the portfolio, an expected value of the portfolio, or something else altogether? Wrapping your numerical results within a string provides meaningful context to the users of your notebooks. Don't assume users will be able to read code. Make your results in notebooks clear to non-technical users. With context, the output of the calculation might display: "The current value of your stock portfolio is: \$23654.84".

Of course, the results of any analytics calculation will vary depending on the data provided to it. In the example of a stock portfolio calculator, the value will change from day to day as stock prices fluctuate or stocks are purchased or sold. Thus, we cannot simply include the value 23,654.84 in a string. Instead, we must embed a variable in the string that contains the calculated value. There are several different ways to embed variables in strings. The examples below show these different techniques.

### Examples of Using Variables within Strings in Python

```
#Variables that you want to add into a string to display to users
name = "Jeff Wall"
age = 21
height = 69.5
```

#Adding variables to a string **using concatenation**. The + sign puts strings together.  
#Notice that context is added to the variables by concatenating several strings together  
#Integers and floats must be wrapped in the built-in str() function to convert them to strings  
#If you fail to wrap numerical values in the str() method, you will receive a Type Error message.  
#You can add two numbers together or two strings together, but you can't add numbers and strings  
`result = name + " is " + str(age) + " years old and is " + str(height) + " inches tall."`

#Adding variables to a string using **f-strings**. The "f" stands for formatted strings.  
#Notice the string starts with the letter "f" before the quotations are added  
#Variables are embedded into the string by wrapping their names in curly brackets (i.e., {})  
`result = f"{name} is {age} years old and is {height} inches tall."`

#Adding variables to a string using the **format()** method.  
#Notice the format() method comes after the quotations are added  
#Variables are embedded into the string by creating placeholders with curly brackets (i.e., {})  
#Then, the format() method accepts variables in the order the placeholders exist in the string  
`result = "{} is {} years old and is {} inches tall.".format(name, age, height)`

#The result variables above would contain: "Jeff Wall is 21 years old and is 69.5 inches tall"  
#The result would change if different values were used for name, age, and height.

With this information about strings, you can now write simple procedural programs that provide more context to the user when performing calculations. We now turn to object-oriented programming.

## Object-oriented Programming

**Object-oriented programming (OOP)** is a paradigm that remains the most widely used in industry for the development of complex information systems. OOP is an extension of procedural programming that adds on extra layers of structure, but also complexity. OOP still relies on functions (a.k.a., methods) to encapsulate business logic. However, functions are further encapsulated within "objects" in OOP. In OOP functions are typically called methods.

OOP can be easier for business users to grasp when they are asked to participate in software development projects, because OOP is founded upon the concept of "objects." In OOP, business logic is

encapsulated within objects. Importantly, many of the objects within information systems stem from knowledge of specific business operations and organizations (i.e., business objects).

An **object** is a digital representation of some "thing"; often something from the real world. A **business object** is a digital representation of an entity or concept that exists within business settings. For example, a shopping cart system needs to digitally represent specific business "things," such as customers, products, and orders. In OOP, a specific customer would be represented by a Customer object, a specific product would be represented by a Product object, etc. In OOP, objects interact with one another to create the functionality of the software system. For example, a shopping cart system might consist of Product objects, Customer objects, ProductInventory objects, SalesPerson objects, Order objects and more to deliver browsing, checkout, and order fulfillment functionality.

When starting a development project, developers should **familiarize themselves with the language of the business domain**. Understanding business language will help you identify important business objects that must exist within the system. To do this well, developers or business analysts who support developers need to understand:

- The **people** who will use the system and their **roles** (both direct and indirect users)
  - Ex. managers and nonmanagers; users in specific job roles and positions
- The **goals** of the people in their job roles
  - Ex. what users are supposed to accomplish in their job positions and why
- Important objects and concepts embedded in **business processes** and **structures** that will be supported by the software.
  - Ex. workflows for specific job tasks; workflows across job tasks; approval structures; products; tools used for work; locations where work occurs; forms used for work

Once you are equipped with a working vocabulary of the business domain, you can begin to decide what business objects (e.g., people, roles, goals, processes, and structures) need to be embedded into the system and how. This is known as functionally decomposing the system (i.e., breaking the system down into its most detailed parts).

## Objects and classes

In OOP, objects are made up of two components: **attributes/properties** and **methods**. Attributes/properties and methods are sometimes called "*members*" of the object.

An object's **attributes/properties** are used to temporarily store data about the object in computer memory. Put differently, an attribute of an object is simply a variable owned by the object that is used to store data about the object. Consider a Customer object and its attributes. The attributes of a Customer object (i.e., variables of the object) might be: firstName, lastName, phoneNumber, and address. An object stores values for each of these attributes (i.e., stored in computer memory as variables) for the specific "thing" that the object represents. For example, a Customer object representing me might have the firstName attribute set to Jeff, the lastName attribute set to Wall, the phoneNumber attribute set to 555-555-5555, and the address attribute set to 123 Fake St.

An object's **methods** represent behaviors and actions that the object can perform, or actions that can be performed on the object. Methods are simply functions that are owned by a particular object. For example, customers might need to change their phone numbers or addresses. So, a Customer object might have methods called: changePhone() and changeAddress(). A Customer object would access logic to change its phone number and address values through these methods. The changePhone() method, for example, might update a field in a database that stores the phone number for the specific customer.

Objects represent specific "things", such as a specific customer, a specific company, or a specific job role. These specific "things" are created from a class that represents the respective group of related "things." Said in a different way, a **class** is a template for creating objects with specific attributes and methods. Classes describe the attributes that an object will possess and how the object's methods will behave. However, classes do not store information about specific "things".

For example, a Customer class would contain the firstName, lastName, phoneNumber, and address attributes described earlier. It would also include the changePhone() and changeAddress() methods with the associated logic for each method. However, the class itself would not store specific information about a particular customer. Again, classes are only templates for creating objects. Objects created from the class "template" store information and enact method behaviors for specific business objects.

To create a class in Python, you simply need to reference the keyword "class" followed by the name of the class and a colon to end the definition of the class. The name of a class should clearly identify what type of business objects will be created from the class (i.e., Customer, Product, Employee). The attributes and methods of the class would be indented on the next line following the colon of the class declaration.

### Syntax for Creating a Simple Class in Python

```
class Customer:  
    #The attributes and methods of the class would go here
```

In the example above, the class declaration is displayed in a **golden color**. This color is used throughout the text anytime a class is referenced to help you differentiate classes from variables and methods.

Objects that represent specific "things" (e.g., a specific Customer) are created by instantiating the respective class (e.g., Customer). **Instantiating** a class simply means that you turn the class template into an object that can store data about a specific instance of the "thing" represented by the class. That was a mouthful. Let's look at an example.

A Customer class could be instantiated into a Customer object to represent a specific customer named Jeff Wall. After creating a class in Python, it can be instantiated by calling the name of the class as depicted below. Multiple objects can be created from the same class. Each object can contain different information, but all of the objects created from the class have the same attributes and methods. Notice in the example below that when an object is instantiated it gets stored in a variable. The first Customer object is stored in a variable named customer1 and the second Customer object is stored in a variable named customer2.

### Example of Instantiating a Class in Python

```
class Customer:  
    #The attributes and methods of the class would go here  
  
#The line below instantiates a new Customer object from the Customer class  
customer1 = Customer()
```

```
#The line below instantiates another new Customer object from the Customer class  
customer2 = Customer()
```

When a class is instantiated, a special method (i.e., function) called the constructor method is called. The **constructor** method of a class tells an object what to do when the object is created through instantiation. Sometimes the constructor of a class is used to set values for the class' attributes or to perform actions that must be completed as soon as the object is created.

In Python, the class constructor method is simply a function named `__init__()` that is contained within the body of the class. **Double underscores** exist before and after the keyword “`init`”. The keyword “`init`” is short for the “initialization” of the object from the class. Even if you do not specify the `__init__()` method, the class always has an implicit `__init__()` method that performs default object construction behaviors. Because an implicit `__init__()` method exists for every class, you only truly need to create the `__init__()` method if you wish to use the constructor to perform specific behaviors when an object is instantiated from a class.

### Example of a Class Constructor in Python

```
class Customer:  
  
    #This is the class constructor method.  
    def __init__(self):  
  
        #class attributes are often initialized within the constructor method  
        #logic you wish to perform when instantiating an object goes here  
  
    #The line below calls the __init__ constructor of the Customer class to create an Customer object  
    customer1 = Customer()  
  
    #The line below calls the __init__ constructor of the Customer class to create another Customer object  
    customer2 = Customer()
```

## Class attributes and methods

In Python, the **attributes** of a class are contained within the constructor method (or sometimes in other methods) of a class. Attributes are simply variables that belong to the class. Each attribute name begins with the keyword “`self`.” `Self` is a reference to the class itself. By appending “`self`” to each attribute, you specify that the attribute variables belong to the entire class. In the example below, the `Customer` class has the following attributes: `firstName`, `lastName`, `phoneNumber`, and `address`. In the example, the constructor of the `Customer` class is used to assign values to the class attributes through its method parameters. In the code below, the parameters `fNameParam`, `lNameParam`, `phoneParam`, and `addressParam` are used to assign values to the `firstName`, `lastName`, `phoneNumber`, and `address` attributes.

### Example of a Class with Attributes in the `__init__()` Method

```
class Customer:
```

```

#This is the class constructor method with several input parameters.
def __init__(self, fNameParam, lNameParam, phoneParam, addressParam):

    #These are the attributes of the Customer class contained inside the __init__() method
    #The values of the parameters above are used to set the values of the class attributes
    self.firstName = fNameParam
    self.lastName = lNameParam
    self.phoneNumber = phoneParam
    self.address = addressParam

#The code below uses the Customer class to instantiate two distinct Customer objects
#Each object is stored in a variable (i.e., customerJeff and customerJen)
#Notice the arguments are passed into the constructor in the order specified in the __init__() method
customerJeff = Customer("Jeff", "Wall", "555-555-5555", "123 Fake St.")

#Notice the arguments are passed into the constructor in the order specified in the __init__ method
customerJen = Customer("Jen", "Wall", "555-555-5556", "123 Real St.")

```

In the example above, the class attributes are displayed with **purple text**. Although attributes are variables, you should understand that there is a distinction between variables and class variables (i.e., parameters).

Class **methods** are simply functions like those discussed in the procedural programming section. Methods accept inputs (i.e., parameters), process the inputs within the body of the method, and return outputs (i.e., return values). In OOP, function signatures are called method signatures. Notice in the method signature below that the first parameter of the method is “self.” In Python, **class methods should almost always have “self” as the first parameter**. Self is a reference to the class the method is contained within.

### Signature for a Class Method in Python

```

class ClassName:

    #The method signature is indented within the class declaration.
    def methodName(self, parameter1Name, parameter2Name, ...parameterNName):

        #The body of the method is indented on the next lines.
        #The logic and the return statement for the method would go here

```

Multiple methods can be added within the body of a class, including the constructor method. In the example below, the class consists of three methods: `__init__()` (i.e., the constructor method), `changePhone()`, and `changeAddress()`. Notice that each method in the example below has “self” as the first parameter. Also notice that the `__init__()` constructor method has several parameters that are used to set the values of the class attributes contained within the `__init__()` method.

### Example of a Class with Attributes and Methods

```

class Customer:

    #This is the class constructor method.
    def __init__(self, fNameP, lNameP, phoneP, addressP):

        #These are the attributes of the Customer class contained inside the __init__ method
        #The parameters above are being used to set the values of the class attributes
        self.firstName = fNameP
        self.lastName = lNameP
        self.phoneNumber = phoneP
        self.address = addressP

    #Below are two methods of the class: changePhone() and changeAddress()
    def changePhone(self, phoneP):

        #logic to change the phone number goes here

    def changeAddress(self, addressP):

        #logic to change the address goes here

#The code below uses the Customer class to create two distinct Customer objects
#Notice the arguments are passed into the constructor in the order specified in the __init__ method
customerJeff = Customer("Jeff", "Wall", "555-555-5555", "123 Fake St.")

#Notice the arguments are passed into the constructor in the order specified in the __init__ method
customerJen = Customer("Jen", "Wall", "555-555-5556", "123 Real St.")

```

Notice in the example above that when an object is instantiated from a class, it is stored in a variable. The first Customer object is stored in a variable named customerJeff and the second Customer object is stored in a variable named customerJen. These names are arbitrary and are only used to help the developer remember what is stored in the variable. The variables could have been called customer1 and customer2, or variableA and variableB.

If you wish to **access the attributes and methods** of a specific object, you must first reference the variable name that the object is stored in. For example, if you wanted to change the phone number of the customer in the first Customer object, you would need to reference the variable customerJeff, because it contains the first Customer object. Then, you would call the changePhone() method of the object in this fashion: `customerJeff.changePhone()`. A dot separates the variable containing the object from the method of the object. Attributes can be accessed in a similar way. First, the name of the variable containing the object must be referenced. Then, the name of the attribute must be referenced in this fashion: `customerJeff.firstName`.

In the example below, two Customer objects are instantiated from a Customer class. The changePhone() method is then called on each object to change the phone number of each Customer object. In the example, the first name of each Customer object is displayed to the user with the two print() commands.

## Accessing the Attributes and Methods of an Object

```
class Customer:  
  
    #This is the class constructor method.  
    def __init__(self, fNameP, lNameP, phoneP, addressP):  
  
        #These are the attributes of the Customer class contained inside the __init__ method  
        #The parameters above are being used to set the values of the class attributes  
        self.firstName = fNameP  
        self.lastName = lNameP  
        self.phoneNumber = phoneP  
        self.address = addressP  
  
    #Below are two methods of the class: changePhone() and changeAddress()  
    def changePhone(self, phoneP):  
  
        self.phoneNumber = phoneP  
  
    def changeAddress(self, addressP):  
  
        self.address = addressP  
  
#Notice the arguments are passed into the constructor in the order specified in the __init__ method  
customerJeff = Customer("Jeff", "Wall", "555-555-5555", "123 Fake St.")  
  
#To change the phone number of customerJeff, you must reference customerJeff and then the method  
#The line below would change the object's phone from 555-555-5555 to 555-555-5554  
customerJeff.changePhone("555-555-5554")  
  
#Notice the arguments are passed into the constructor in the order specified in the __init__ method  
customerJen = Customer("Jen", "Wall", "555-555-5556", "123 Real St.")  
  
#Because customerJen is a different object, its phone can be changed independent of the other object  
#The line below would change the object's phone from 555-555-5556 to 555-555-5557  
customerJen.changePhone("555-555-5557")  
  
#Object attribute values can be access by referencing the attribute name you wish to retrieve  
print(customerJeff.firstName) #this would display "Jeff" to the user  
print(customerJen.firstName) #this would display "Jen" to the user
```

With the information you have to this point, you should be able to create a class, instantiate objects from a class into a variable, and use the attributes and methods of an object stored within a variable. Later chapters will show you how to create more complex functionality by pairing behaviors of multiple different classes.

## Variable scope in OOP

Variables also possess a scope in OOP. In OOP, variables are used to create class attributes, method parameters, and are used within the body of class methods. In Python, variables can also exist outside of classes. This is not the case for other languages that are strictly object-oriented, like Java. In Java, all

variables exist inside of classes. Python can be written in procedural form, object-oriented form, or more typically in a hybrid form.

The **scope of a class attribute variable** can be adjusted. By default, class attributes in Python are publicly available and can be accessed anywhere, including within the class itself (i.e., by members of the class), within other classes (i.e., by members of other classes), and outside of the class.

Within the class, a class attribute is often referenced by using the keyword “self” followed by the name of the attribute: `self.attributeName`. The example below shows the `phoneNumber` attribute being created in the `__init__()` method and then referenced within the `displayPhone()` method of the same Customer class by referencing `self.phoneNumber`.

### Accessing a Class Attribute within a Class

```
class Customer:  
  
    def __init__(self, fNameP, lNameP, phoneP, addressP):  
  
        #These are the attributes of the Customer class contained inside the __init__ method  
        #The parameters above are being used to set the values of the attributes  
        self.firstName = fNameP  
        self.lastName = lNameP  
        self.phoneNumber = phoneP  
        self.address = addressP  
  
    #In the method below the phoneNumber from the __init__() method is referenced.  
    def displayPhone(self):  
  
        #referencing the self.phoneNumber attribute within the class is okay  
        print(self.phoneNumber)
```

When accessed outside of a class or within a different class, the name of the variable containing the object must be referenced first and then the name of the object’s attribute: `object.attribute`. In the example below, a Customer object is instantiated from the Customer class. The object exists outside of the Customer class. The object is stored in a variable named `customerJeff`. The `phoneNumber` attribute is accessed by first referencing the variable with the Customer object: `customerJeff.phoneNumber`.

### Accessing a Public Class Attribute outside a Class

```
class Customer:  
  
    def __init__(self, fNameP, lNameP, phoneP, addressP):  
  
        #These are the attributes of the Customer class contained inside the __init__ method  
        #The parameters above are being used to set the values of the attributes  
        self.firstName = fNameP  
        self.lastName = lNameP
```

```

    self.phoneNumber = phoneP
    self.address = addressP

#customerJeff exists outside of the Customer class and is an object instantiated from the class
customerJeff = Customer("Jeff", "Wall", "555-555-5555", "123 Fake St.")

#The phoneNumber attribute of the object can be accessed through the variable containing the object
#This is okay for publicly available class attributes
print(customerJeff.phoneNumber)

```

In some programming languages, like Java, class attributes are rarely publicly available. The attributes are made private so that they can only be accessed from within the class that owns them. Python also allows for the creation of **private class attributes**. To make a class attribute private, the name of the class attribute variable must start with two underscores. For example, a private phoneNumber attribute would be named: self.\_\_phoneNumber. Private class attributes cannot be accessed outside of the class, but can be accessed by methods inside of the class where it was initialized.

### Accessing a Private Class Attribute within a Class

```

class Customer:

    def __init__(self, fNameP, lNameP, phoneP, addressP):
        #The phoneNumber attribute is private. All other attributes are public
        self.firstName = fNameP
        self.lastName = lNameP
        self.__phoneNumber = phoneP
        self.address = addressP

    #In the method below the __phoneNumber from the __init__() method is referenced.
    def displayPhone(self):
        #referencing the private self.__phoneNumber attribute within the class is okay
        print(self.__phoneNumber)

#customerJeff exists outside of the Customer class and is an object instantiated from the class
customerJeff = Customer("Jeff", "Wall", "555-555-5555", "123 Fake St.")

#The firstName attribute of the object can be accessed through the variable containing the object
#This is okay for publicly available class attributes
print(customerJeff.firstName)

#The line below would throw an error because private attributes can't be accessed outside of the
#class
print(customerJeff.__phoneNumber)

```

The variable scope rules that apply to other types of variables in procedural programming also apply in object-oriented programming. For example, method parameters (i.e., function parameters) can only be

accessed within the body of the method that owns the parameters. Variables inside of a method can only be accessed within that method. Variables created outside of a class can be accessed anywhere, including outside of the class and inside any method within any class. Refer back to the procedural programming scope examples if you need to recall these rules.

## Getter and setter methods

As stated earlier, class attributes can be made private in Python to protect the data stored within them from being changed by code outside of the class. Sometimes, you will need to specify conditions for whether to set the value of an attribute or not. This can be accomplished through special methods.

Methods that set the value of a private class attribute are called **setters**. Methods that get the value of a private class attribute are called **getters**. Because getter and setter methods are members of the same class as the private attributes, they can access and alter the values of the private attributes. In Python, getter and setter methods are not required unless you designate an attribute as private. Each organization has different rules for the use of public and private attributes. Follow the norms of your organization.

A getter method returns the value stored in a private class attribute and generally does not possess parameters (except for the required self parameter). A setter method generally does not return anything and should have at least one parameter to represent the new value that will be stored in the respective private attribute. Constructor methods are often used as setters to set values for all class attributes when an object is first instantiated.

### Example of Getter and Setter Methods in Python

```
class Customer:  
    #fNameP and lNameP are used to set values for the __firstName and __lastName attributes  
    def __init__(self, fNameP, lNameP):  
        #The firstName and lastName attributes are private.  
        self.__firstName = fNameP  
        self.__lastName = lNameP  
  
    #a getter method for the private firstName attribute  
    def getFirstName(self):  
        return self.__firstName  
  
    #a setter method for the private firstName attribute  
    def setFirstName(self, fNameP):  
        #set the value of the firstName attribute to the value stored in the fNameP parameter  
        self.__firstName = fNameP  
  
    def getLastname(self):  
        return self.__lastName
```

```

def setLastName(self, lNameP):
    self.__lastName = lNameP

#Below, the constructor is used to set initial values for the private attributes
customerJeff = Customer("Jeff", "Wall")

#Using a getter method to print out the private first name attribute of the customerJeff object
print(customerJeff.getFirstName()) #would display "Jeff"

print(customerJeff.firstName) #would throw an Error because firstName is private

#Using a setter method to change the first name of the private firstName attribute from "Jeff" to "John"
customerJeff.setFirstName("John")

print(customerJeff.getFirstName()) #would now display "John"

customerJeff.firstName = "Jamie" #would throw an Error because firstName is private

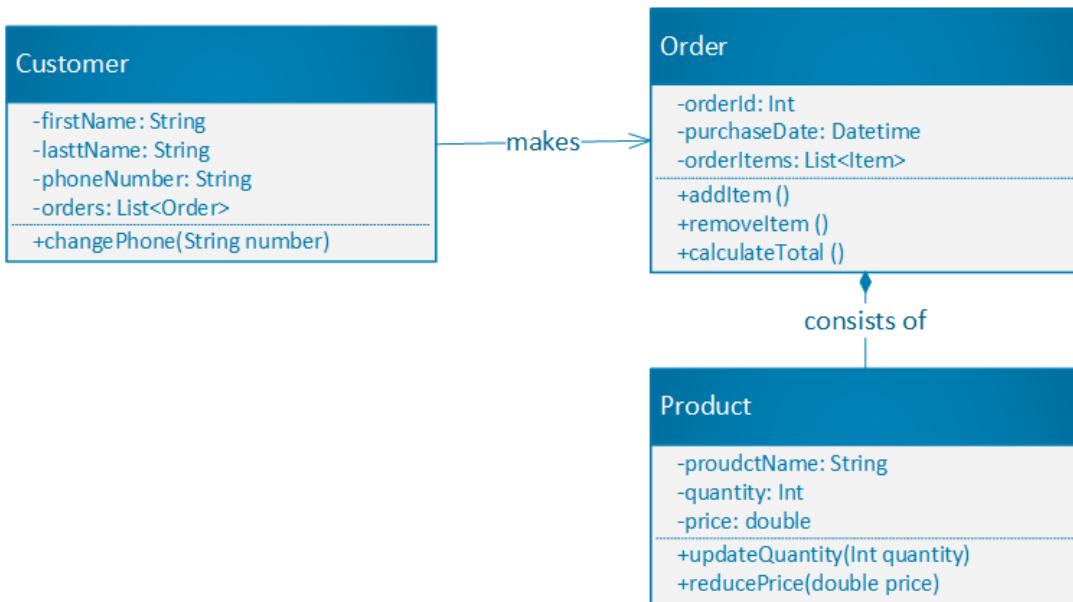
```

## Class diagrams

System and business analysts often produce diagrams called class diagrams to help visualize OOP programming code that will need to be written to develop an object-oriented system. There are many different types of class diagrams. For example, a **domain class diagram** depicts the core business objects (e.g., customers, products, transactions, accounts, employees, departments, etc.) that a system needs to keep track of. The diagram also keeps track of the attributes of each class. Class diagrams also depict relationships between the core business objects (e.g., customers purchase products, departments consist of multiple employees, etc.). Domain class diagrams are similar to entity relationship diagrams used to model databases that store business data.

Another type of class diagram is a **design class diagram**. Design class diagrams capture relationships between classes that can be represented in code. Design class diagrams still depict the attributes of each class. However, they also capture the methods (i.e., behaviors) of each class. Figure 2.1 shows a simple design class diagram. The diagram shows that the Customer class has an association with the Order class, such that Customers make Orders. The diagram also shows that an Order is composed of Products. The members (i.e., attributes and methods) of each class are also depicted. For example, the Customer class has first name and last name attributes, and the Order class has a purchase date attribute. The classes in the diagram also possess behaviors in the form of methods, such as the addItem() method in Order that would add a Product to the Order. A line is often added to a class diagram to differentiate the class's attributes from its methods as shown in the diagram below.

**Figure 2.1: Simple Example of a Design Class Diagram**



Class diagrams will be used throughout this text to help you develop the ability to understand the connection between class diagrams and the code developers might write to implement the diagrammed system. Business analysts or developers often create class diagrams to help developers build complex software.

You now possess fundamental knowledge about how to write code in procedural and object-oriented form. We will continue to build onto these fundamental skills in later chapters. It is okay if you don't feel perfectly percent comfortable with the concepts in this chapter. Be patient and put in the time to read and try out the topics covered in each chapter. Like most learning, with time and repetition, the concepts will sink in. Each chapter builds on concepts in previous chapters, so you will see these programming concepts again.

## Inheritance in OOP

An important feature of OOP is inheritance. **Inheritance** is concerned with hierarchies of objects. Inheritance allows a child class (aka, subclass) to inherit properties of a parent class (aka, superclass). Let's use a biological example you are probably familiar with to demonstrate how inheritance works.

Biologists rely on biological classification to classify species and identify new species. Let's explore the Mammalia class. Mammals have certain attributes and methods (i.e., behaviors). For example, mammals exhibit common attributes, such as having hair and mammary glands. They also exhibit common behaviors, such as giving birth to live offspring. Many creatures are mammals. For example, dogs, cats, and humans are all classified as mammals. However, it isn't true to say the opposite. One way to check whether you are dealing with an inheritance hierarchy is to make hierarchical statements about perceived parent-child relationships. For example, it is true to say that all dogs are mammals. However, it is not true to say that all mammals are dogs. Therefore, the Mammal-Dog relationship has a reasonable inheritance structure in which Mammal is the superclass and Dog is a subclass. Inheritance relationships are sometimes called "**is a**" relationships because the subclass "is a" variant of the superclass (e.g., a Dog "is a" Mammal).

This means that a Dog class would inherit hair, mammary glands, and giving birth to live offspring from the Mammal class. In the pseudo-code below, the Dog class inherits the two attributes and the one

method of Mammal. Notice also in the example that subclasses can have attributes and methods that the superclass does not. The Dog class has one other attribute (e.g., hasFloppyEars) and one additional method (e.g., bark()) that the Mammal class does not. Figure 2.2 shows the class diagram representation of an inheritance relationship. Notice in the diagram that multiple levels of inheritance exist. Knowing that attributes and methods are inherited, what are the attributes and methods of the Dog class?

### Inheritance in Python

In Python, a class can inherit attributes and methods from multiple parent classes. This is referred to as **multiple inheritance**. The Python syntax for inheritance and multiple inheritance is presented below.

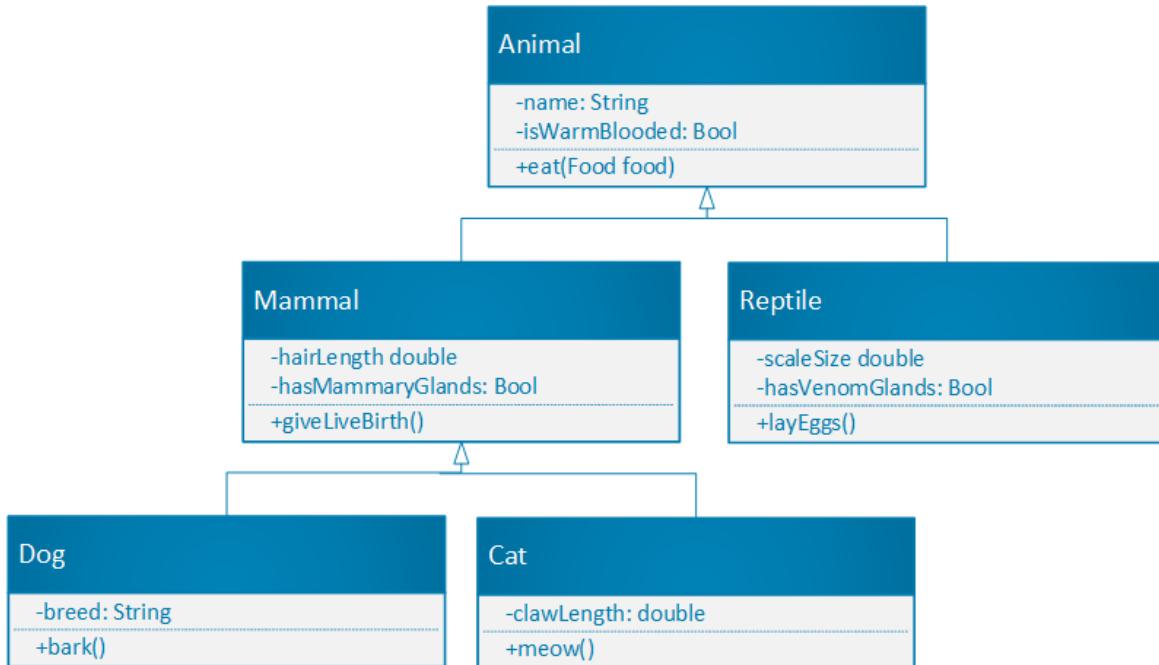
Parent classes are contained inside parentheses in Python syntax

```
#The Dog class will inherit attributes and methods from the Mammal class  
class Dog(Mammal):
```

Multiple parent classes can be called within the parentheses with a comma separating them

```
#The Cyborg class will inherit attributes and methods from both the Human and Robot classes  
class Cyborg(Human, Robot):
```

Figure 2.2: Example of Inheritance in a Class Diagram



The code below shows an example of inheritance in Python.

### Simplified Code Showing An Example of Inheritance

```
class Mammal:  
    def __init__(self):  
  
        self.hasMammaryGlands = True  
        self.hairLength = None  
  
    def giveBirth(self):  
  
        #code for live birth would go here  
  
class Dog(Mammal):  
  
    #because Dog inherits from Mammal, it will automatically possess the giveBirth() method  
    #it will also possess the attributes: hasMammaryGlands and hairLength  
  
    #sub classes can have their own unique attributes and methods that the super class doesn't  
    def __init__(self):  
  
        self.hasFloppyEars = True  
  
    def bark(self):  
  
        #code for barking would go here
```

Sometimes, a subclass needs to inherit some of the methods of a superclass, but also needs custom implementations for other inherited methods. For example, the duck-billed platypus does not give live birth like most other mammals do. Instead, the duck-billed platypus lays eggs. So, a DuckBilledPlatypus class would inherit from the Mammal class because it is a mammal, but it would also need to override the Mammal class implementation of the giveBirth() method. **Overriding** is the act of ignoring a method implementation from the parent class to create a custom implementation of the inherited method in the child class. The code below shows how methods are overridden to produce different behavior for a subclass.

### Inheritance with Overriding

```
class Mammal:  
    def __init__(self):  
  
        self.hasMammaryGlands = True  
        self.hairLength = None  
  
    def giveBirth(self):  
  
        #code for live birth would go here  
  
class DuckBilledPlatypus(Mammal):
```

```

#sub classes can have their own unique attributes that don't exist in the super class

def __init__(self):
    self.hasPoisonGlands = True

#the method below tells the program to ignore the giveBirth() method in the Mammal class
def giveBirth(self):
    #code for egg laying goes here instead of accepting live birth from Mammal

```

## **Composition in OOP**

Inheritance allows child classes to share the structure of parent classes. Although inheriting the structure of another class can be useful, it creates tight links between parent and child classes. In programming, a tight link between classes is called **tight coupling**. Tight coupling makes large systems hard to change and easy to break. In a tightly coupled system, one small change to a class can have a ripple effect on other classes that are tightly coupled to the class that was changed. Inheritance also forces a child class to adopt all of the behaviors of the parent class, unless overridden, even if the behaviors are not needed by the child class.

Many developers prefer to use the concept of composition to borrow functionality from other classes rather than inheritance. **Composition** allows a class to gain the behaviors of another class through what is called delegation. **Delegation** is the act of assigning a behavior within a class to be performed by another class. Whereas inheritance is an “**is a**” relationship, composition is a “**has a**” relationship. For example, an Employee class could borrow functionality from a Contact class such that the Employee class has access to the attributes (e.g., address and phone number) and methods (e.g., get and set address) of the Contact class. Thus, the Employee class would have a Contact object instantiated from the Contact class.

Composition can be accomplished in many different ways. Composition does not necessarily solve the tight coupling problem. However, if done properly, it can create more loosely coupled systems that are less prone to breaking down when changes are made to the system. We will explore tightly and loosely coupled system components in later chapters. For now, just recognize that the behaviors of a class can be composed from multiple other classes through delegation rather than through inheritance. The relationship between Product and Order in Figure 2.1 shows an example of composition. In that example, an Order was composed of Product objects. Further, the Customer class was composed of Order objects. The components that a class gains access to through composition are often stored in the class attributes of the class that use the components. Thus, in Figure 2.1, the Customer class has an attribute called orders that stores Order objects and the Order class has an attribute called orderItems to store Product objects.

Although interfaces can be used to compose a class with different behaviors, it is better to pass or embed delegate classes into a class. This is often accomplished by passing delegate classes into a constructor method or another method. Passing a class into a constructor or method is a special type of composition

called **aggregation**. Although you can also embed a delegate class into another class without passing it in (i.e., true composition), recognize that this practice can lead to tight coupling.

## Chapter 2 Assignment

Please see the [Chapter Assignments Workbook](#) for the Chapter assignment.

# Chapter 3:

## Debugging, AI Code Assistance, Software Testing, and Pair Programming

Depending on your background and experience, you may have found the assignment in Chapter 2 to be too difficult to complete, or you may have completed it but it didn't run properly. Maybe you ran into some "errors" that you didn't know how to fix alone. Maybe you even conducted Google searches that led to appropriate fixes, but the lingo in the forums or blog posts was too foreign to understand. Even if you are a budding computer scientist and found the assignment to be easy, if you become a software developer, you or your team will eventually run into "errors". As a business professional, you may never choose to write a program yourself. However, understanding the difficulties of programming will help you to empathize with software developers that you may lead or work with, and will help you respect the time and costs associated with software development. A great deal of developing software is finding and correcting errors, sometimes called "bugs".

This chapter will introduce you to some tools, tricks, and practices to systematically find errors or avoid them as much as possible. It will also help you learn how to test your code for quality assurance purposes. Most information systems consist of complex software applications. When you add a new feature to a complex software system or make a simple change to improve an existing feature, you can inadvertently break other features of the software system. Testing your software ensures that your code works as expected and that changes you make to software don't create unintended consequences.

### Be Methodical and Patient

First, a word of advice: be slow and methodical as you are learning to program. Most mistakes made by first time developers are related to syntax errors (e.g., forgetting a colon, misspelling a variable or function name, etc.). For the next few assignments, write a line of code and then examine it carefully for syntax errors. Your time is limited, but it can take longer to find and fix an error than to be methodical. Try to prevent as many syntax errors as possible. As your skills develop, syntax will feel more natural.

Also remember to be patient. Be patient with the process of learning a new language and a new way of thinking. Be patient with yourself. Programming doesn't come easily to everyone. If that happens to be true for you, don't worry. Some very excellent programmers started out writing "buggy" and error prone programs. Many large successful tech companies, such as Facebook, also started with poorly written code.

Finally, be patient with others. Even if you think you are a good programmer and plan to be a software engineer, chances are you will learn how much you don't know when you start your first full-time development job. After five years on the job, you will look back on the first programs you wrote and have a hearty laugh with yourself and your colleagues. If you are developing complex system architectures right after finishing your undergraduate degree, you are the exception and not the norm. Most system architects that handle the design of truly complex systems do not receive their architectural job positions

for 15 to 20+ years into their careers. Hopefully, your mentors in your first job will be patient with you. Learn to develop that patience now so that you can be a good manager and mentor later.

If you find programming to be difficult and plan to avoid jobs that require programming knowledge, let this experience teach you how to approach difficult topics and how to seek help when you need it. Seek out mentors among your peers if you are struggling. You will likely need to find a mentor in the future to move your career forward. It is good to practice seeking assistance now. Asking for help doesn't come easily to everyone. Similarly, providing help doesn't come naturally for some. Work on being both a mentor and mentee.

## Errors and Exceptions

In Python and other languages, there are different types of "errors" that can occur in a program. The big distinction in many languages is between Errors and Exceptions.

### Errors

True errors will stop the execution of your program without being able to recover from the error. Several **Errors** in Python are typically out of your control as a developer, and are caused by the environment (e.g., server) on which your code runs. For example, servers (i.e., powerful computers) have limited amounts of memory. Remember that your program stores data in variables. Behind the scenes, the data in those variables is stored in memory on a server. If your program loads too much information into memory (e.g., loads a big file into memory), or memory is being used by other programs running on the same server, your program may stop executing and throw an Error. You can't always recover from such errors, because you may not have control over the server and how memory is allocated.

Most Errors like memory errors are handled by the operations (Ops) team in the IT department that maintains programs while they run on the organization's servers. In the case of memory concerns, you can try to build in some protections as a developer, such as checking the size of data files you intend to load into memory before actually loading them. For example, you might only allow files to be loaded into memory if they are below a certain size (e.g., below 100MB). However as a developer, you may not be able to control perfectly for errors like memory errors.

Another Error that you can control that will also kill the execution of your program are **syntax errors**. Syntax errors are often caused by poor programming practice, such as forgetting indentations, colons, etc. In Python for example, syntax errors such as failing to add a colon after a method signature or forgetting to add the "def" keyword at the beginning of a method signature will cause an error to be thrown when you run the program. Most IDEs have a console window that will show you these and other errors when you run your code. In the Jupyter IDEs, each cell acts as its own console. Errors that occur within a cell will be reported in the output of the cell when you run the cell.

### Exceptions

**Exceptions** are "errors" that occur and can be caught and handled by your program without shutting down the program. **Catching an exception** simply means that your program doesn't necessarily stop executing when an exception occurs. Catching an exception allows you to redirect the flow of the code. Depending on how bad an Exception is or what it affects are, you can sometimes catch an Exception and

still provide reasonable amounts of functionality to the user. At minimum, you can provide users with a meaningful message about what occurred and how they can fix the problem, or provide them with a message letting them know that the issue is being looked into.

Some languages like Java differentiate between two types of Exceptions, checked exceptions and unchecked exceptions. Python only has the equivalent of unchecked Exceptions. **Checked exceptions** in Java are exceptions that occur at runtime (i.e., when the program is running) and are known to the Java compiler at compile-time (i.e., before the program has run). Like errors, checked exceptions are often caused by the environment, such as by files, databases, and networks that your program uses to accomplish tasks. However, checked exceptions do not terminate your program the same way that errors do. For example, if a database connection throws an Exception because the database isn't working, you could catch the Exception and tell the program to try connecting to a different database. Alternatively, you could provide the user with a message telling them that the database is down and that personnel are working on the problem.

**Unchecked exceptions** happen at runtime and are not known to the compiler at compile-time. You don't need to fully understand compiling to write programs. Just recognize that your code has to be compiled (i.e., converted) from what you write to something your computer understands. If you are interested in learning more about compiling, you can Google the topic or read a computer science book on the topic.

All Exceptions in Python are the equivalent of unchecked exceptions, because Python is an interpreted language that doesn't pre-compile the code you write like Java and some other languages do. Thus, it does not check for possible exceptions before you run your code. You won't know about Python errors until you run your code. If you want to learn the difference between compilation and interpretation, you can also Google that topic. You don't need to fully understand the distinction to write useful code though.

Exceptions are often caused by the way you write a program, such as failing to consider the strange ways that users might use your program. If you aren't careful to check the information users provide to your program, **user input can be the cause of many Exceptions**. User input should NEVER be fully trusted.

Many unchecked exceptions are thrown because developers fail to account for the type of information a user might provide to the system. For example, if your program divides two numbers, a ZeroDivisionError could be thrown if the number in the denominator is zero (i.e., you can't divide by zero). You might also receive an error if a user inputs text into a variable intended to be a number. If mathematical operations are attempted on strings, an exception will be thrown. Because Python is a dynamically typed language, it is possible for a user to pass text into a variable intended to be used for numbers. Dynamically typed languages require that you pay extra close attention to user inputs to avoid exceptions.

If a piece of information that a user provides is not particularly important and all other information they provided is correct, you can catch an Exception and continue to execute the rest of the program to process the valid information. In such cases, you would want to provide a message to the user describing which parts of the program succeeded and which parts failed. However, if the information or action related to the Exception is central to the functionality of a program, you might not want to continue the execution of the program. You would then display a message to the user explaining the issue without executing other parts of the program.

**Be careful when writing error messages to users.** For example, if your message tells a user that a feature didn't work and that they should try again, the user might try again and again and again. If the

program uses a network to call a database or a program on another server, these requests will be executed over and over again. If an error message like this is received by many users at the same time and they all try to access the feature over and over again, they may overload a server with their repeated requests. It is often safer to let users know that an error has occurred and that it is being worked on.

## Syntax for catching exceptions

There are several ways to catch and handle Exceptions in Python. The first is to **catch exceptions in the methods (i.e., functions)** in which an exception occurs. This is accomplished through try-except and try-except-finally statements. First, you place the code you want to perform into the try block of the try-except statement. For example, assume that you want to read the contents of a file from your hard drive into your program. This could be accomplished by using the built in open() method in Python. The core operation you wish to perform (e.g., read a file) would be contained in the **try block**.

### Try Block in Python

```
try:  
    #code to try goes here
```

If code in the try block throws an Exception, such as when a file doesn't exist, the flow of control would move from the try block to the **except block**. The except block requires that you specify the type of Exception that will be thrown through a parameter that stores information about the Exception, such as the stack trace. The **stack trace** keeps track of all of the classes and methods called in your program in the order that they were called. The stack trace can help you diagnose where a failure is occurring in your program. The stack trace and simpler error messages are handled through the except block. In the example below, if the code fails to run, the except block receives the Exception and prints the Exception's built-in error message to the user.

### Try-Except Statements

```
try:  
    #try code goes here  
except Exception as e:  
    #code to handle exception goes here, such as printing a message  
    #the variable e stores information about the exception  
    print(e)
```

The **finally block** is executed after the try and except blocks. Anything within the finally block executes whether or not an Exception occurs. So, if no Exception occurs, the flow of control will move from the try block to the finally block. If an Exception occurs, the flow of control will move from the try block, to the except block, and then to the finally block.

The finally block is optional. Although some developers suggest that you should always have a finally block to return outputs, others rarely use the finally block. Python also has a distinct **else block** that other languages do not, which executes code only when the try block succeeds. If an Exception occurs, the else block does not execute.

### Try-Except-Else-Finally Statements

```
try:  
    #try code goes here  
except Exception as e:  
    #code to handle exception goes here, such as printing a message  
    print(e)  
else:  
    #code to run only if an exception does NOT occur goes here  
finally:  
    #code to run regardless of whether an exception occurs or not goes here
```

The examples below show how an Exception might be handled in Python if you wanted to read a file from your hard drive into memory to display the file contents for a user.

### Example of a Try-Except-Finally Statement to Catch Exceptions

```
class FileManager:  
    #The readFile() method is contained in a class called FileManager  
  
    def readFile(self, fileLocation):  
  
        #The readFile() method accepts a file location from the user and tries to open the file.  
  
        try:  
  
            #Try to read a file specified in the fileLocation parameter with the open()  
            #method  
            #the second parameter of open is the mode. The argument "r" means read  
            #mode  
            #read mode reads the contents of the file into the variable "file"  
            file = open(fileLocation, "r")  
  
            #The FileNotFoundError is a built-in Exception that Python can catch.  
  
        except FileNotFoundError as e:  
  
            #if the file isn't found, an exception is thrown and the error is printed out  
            #e is the variable that stores information about the Exception  
            print(e)  
  
        finally:  
  
            #this line is run whether the try block succeeds or fails  
            print("this is the end of the readFile() method")
```

Although there is more that can be done with Exceptions, such as creating multiple except blocks for different possible Exceptions that could be thrown and catching from a list of different Exceptions, you now have enough knowledge to ensure that can account for unintended uses of your software program.

Python Exceptions are fairly well documented online. However, if you don't know which of the many Exception classes you should catch in your except block, then you can run your code with values that should produce an Exception. Look to the console for the particular Exception that is thrown. You can then add the appropriate Exception type to the expect block. Python and other languages also possess a generic Exception called Exception that you can catch. However, it is better to catch specific Exceptions than a generic Exception so that you can provide meaningful error messages to users.

## Propagating exceptions

Another way to catch exceptions is to **catch them at their callers**. The “caller” is the line of code that calls a method/function. To use this approach, you wouldn’t catch the Exception in the method/function causing the Exception, but in the part of the program that calls the failing method/function. Catching exceptions at the method caller can be accomplished in many programming languages, including Python. Catching an error in this way is sometimes referred to as Error/Exception **propagation**, because the Error/Exception propagates down the stack of method calls from one method to another. In Python, you simply move the try-except blocks from the offending method to the location where the offending method is called. Let’s examine an example to further clarify how propagation works.

### Example of Error Propagation

In the code below, the callerMethod() of the Caller object calls the readFile() method of the FileManager object. In this example, callerMethod() is the caller method. The try-except statement exists within the caller method instead of in the readFile() method where the Exception actually occurs. Thus, the Exception propagates from the readFile() method to the callerMethod(). In the earlier example, the try-except statement existed inside the readFile() method instead of in the callerMethod().

```
class Caller:  
    #callerMethod() is the caller of FileManager.readFile()  
    #the error in FileManager.readFile() is passed to callerMethod() and caught in callerMethod()  
    def callerMethod(self):  
        #notice that the try-except blocks are here instead of in the readFile() method below  
        try:  
            #A FileManager object is instantiated from the FileManager class below  
            fileManager = FileManager()  
  
            #The readFile() method of the FileManager is called with a file location  
            fileManager.readFile("/home/jeff/home.txt")  
  
        except FileNotFoundError as e:  
            print(e)  
  
    #the class below could exist in the same cell or in a different cell in the Jupyter IDEs  
  
    class FileManager:
```

```
#Notice that there is not a try-except block in this method like in the previous example  
#Any errors that occur during the file reading process will be propagated to callerMethod()  
  
def readFile(self, fileLocation):  
  
    file = open(fileLocation, "r")
```

### Throwing exceptions on command

It is also possible to throw an Exception at any point in your program on your command. Sometimes, you know an error is going to occur simply by looking at user inputs. For example, if a user passes zero into a parameter that is intended to be used in the denominator of a division operation, you know that an error will occur. You cannot divide by zero. Now imagine that you have a complex set of computations you must perform to make a business decision. Assume that the computations are so complex that it takes 2-3 minutes or longer to finish the computation. If the division operation is one of the last operations in the complex calculation, you might leave a user waiting for 2-3 minutes or more before an Exception is thrown. That is a painful experience for a user. Instead, if you know an error is going to occur, you can throw the respective Exception before trying to execute time consuming calculations. This practice allows you to immediately advise the user to change their inputs. With this practice, no waiting happens for the user.

In Python, this is accomplished by using the "raise" keyword followed by the name of the Exception you wish to throw: **raise Exception()** or **raise FileNotFoundError()**. See a more complete example of throwing an Exception below.

#### Example of Throwing an Exception on Command

The code below shows a FileNotFoundError being thrown if the file location does not have a specific filename. The "!=" character means "not equal to" in Python. We will learn more about if-else() statements in another chapter. Don't worry about that syntax now. Just recognize that the program will check that the appropriate file is called. If an inappropriate filename is provided, an Exception will be thrown. If the appropriate filename is provided, it will then be opened.

Notice that an argument is passed to the FileNotFoundError() method that reads, "You only have rights to access apple.txt". Using this format, you can pass custom messages to an Exception. This custom message gets returned through the Exception. In the example, the thrown error would be propagated to the caller method because the readFile() method does not include a try-except statement as seen in the prior example.

```
class FileManager:  
  
    def readFile(self, fileLocation):  
  
        if(fileLocation != "/home/jeff/apple.txt"):
```

```
        raise FileNotFoundError("You only have rights to access apple.txt")  
  
    else:  
  
        file = open(fileLocation, "r")
```

## ***Creating custom exceptions***

Finally, you should know that you can create your own custom Exceptions in Python. This can be done by inheriting from the Exception class (e.g., creating a subclass of Exception). Let's create an Exception called NotAppleTextFileException. We will use this Exception instead of the FileNotFoundError(). We will throw this Exception when the file location is not equal to "/home/jeff/apple.txt" as in the previous example.

Although throwing the NotAppleTextFileException is a little absurd, it is more correct than throwing a FileNotFoundError, such as in the previous example. Custom Exceptions allow you to provide more detailed feedback to users and provide developers with non-standard Exceptions that fit proprietary system functionality. If you can't find a built-in Exception to catch errors that may occur within your program, build your own custom Exception classes and throw them to catch your specialized "errors."

### **Custom Exceptions**

To create a custom Exception, you can either create a default value for the message as a parameter in the `__init__` constructor method, or hard code the message and not accept a "message" parameter to `__init__`. Notice that the example below sets a default message.

```
class NotAppleTextFileException(Exception):  
  
    def __init__(self, message="You must call the apple.txt file for the program to work."):   
        Exception.__init__(self, message)  
  
class FileManager:  
  
    def readFile(self, fileLocation):  
  
        if(fileLocation != "/home/jeff/apple.txt"):  
  
            raise NotAppleTextFileException()  
  
        else:  
  
            file = open(fileLocation, "r")  
  
#Instantiating the FileManager class and calling readFile() with an incorrect file name  
#This code would throw the NotAppleTextFileException()
```

```
fileManager = FileManager()  
fileManager.readFile("text.txt")
```

You now have a working understanding of how to use Exceptions in Python. From a managerial perspective, you must recognize the importance of building user-friendly and high-quality systems. The way you handle exceptions is an important part of building usable and quality systems. Users will misuse systems, databases will go down from time to time, files will not be found, and other errors will occur in software. You need to think about and plan for possible exceptions to the normal flow of control within a program.

## Debugging

Sometimes your program will execute with no Errors or Exceptions, but will not provide the result that you expect. Other times, you will want to understand why an Exception was thrown because you aren't sure why your program isn't working correctly. Debugging is the act of finding and fixing errors in your code. When you have complex programs, it isn't always easy to find where problems exist. However, many IDEs possess tools to help you debug programs. There are also some tricks to make debugging easier. To begin, you need to understand a little about control flow in programs.

## Control flow

**Control flow (aka., flow of control)** refers to the order in which statements and methods are executed in a program. We will learn more about control flow in later chapters when we discuss if-then-else logic and loops. For now, let's explore a simple example of flow control using methods in the code example that follows.

In the code, you will see two classes, the CustomerManager class and the Customer class. In the Jupyter IDEs, you might include each of the classes in its own cell. Sometimes you don't want classes to have the ability to control themselves, such as creating and destroying themselves. To solve this issue, you might create a Manager or Controller class. The **Manager or Controller class** deals with creating, using, and destroying the objects they manage/control. So the CustomerManager class in the example below might be used to manage the creation and use of Customer objects.

Programs are executed line by line, but if calls are made to methods in other classes, the line by line flow jumps from one class to another in a systematic, linear fashion. Let's take a look at the flow of control in the code below. See if you can figure out the flow of control before reading the paragraph after the code example.

1	import uuid #A Python library that creates id numbers
2	class CustomerManager:
3	def createCustomer(self, nameP):
4	customerObject = Customer() #instantiate a Customer object

```

5      customerObject.setName(nameP) #set the name of the Customer object
6
7      return customerObject
8
9  class Customer:
10
11     def __init__(self):
12
13         self.name = None #set up the name attribute of the Customer
14
15         self.id = uuid.uuid4() #set the id attribute with a random id using the uuid library
16
17     #a setter method for the Customer name attribute
18     def setName(self, nameP):
19
20         self.name = nameP
21
22     #a getter method for the Customer id attribute
23     def getCustomerId(self):
24
25         return self.id
26
27 customerManager = CustomerManager()
28
29 newCustomer = customerManager.createCustomer("Jamie Doe")

```

The program will start off on **line 15** by creating a new CustomerManager object and storing it in the customerManager variable. Remember that classes (i.e., the CustomerManager and Customer classes in the example above) do not run until they are instantiated. Line 15 instantiates a customerManager object. When calling CustomerManager(), the program will jump to the CustomerManager class and execute the constructor (i.e., the `__init__()` method). In the code above, there is no explicit constructor method in the CustomerManager class. Recall that all classes have constructors that perform default behaviors whether you explicitly create an `__init__()` method or not. Once the constructor has completed its object instantiation work, the program will jump to **line 16**. Line 16 calls the `createCustomer()` method of the CustomerManagement object created on line 15. This method call will jump the flow of control to **line 3** of the CustomerManager class to execute the `createCustomer()` method.

On line 4, a new Customer() object is created, which would jump the flow of control to the constructor method of the Customer class on **lines 8-10** to create a randomly generated id for the customer. The program would then jump back to where it left off in the CustomerManager class and move to **line 5**. Line 5 would jump the flow of control to the Customer class again to the `setName()` method on **lines 11-12** to set the name attribute of the Customer object. The code would then move back to the CustomerManager class on **line 6**, which would return a Customer object to **line 16** to be stored in the newCustomer variable.

As this example shows, understanding the control flow of a program can become complex as the number of classes and methods increase. This is just a simple example. Real systems consist of tens to thousands of classes or functions that work together to provide an experience for users. The important thing to recognize is that flow starts at a designated spot in the program and continues line by line until a

method call is encountered. The program then jumps to that method, moves line by line through that method and starts back where the method call was made. The flow of control will become more clear as you experiment with breakpoints.

## **Breakpoints**

**Breakpoints** are a feature built into many IDEs that allow you to pause the execution of your program at desired lines in your code so that you can explore the value of variables and other aspects of your program at those specific points in the flow of control. When debugging with breakpoints, the program will pause at each breakpoint you create. The program will stop at each breakpoint until the program throws an error or until it has gone through all breakpoints and is finished executing the program.

As you are learning about flow control, it can be helpful to set breakpoints on every line of code in every class and/or function within your program so that you can see how the program jumps from one method to another and then back again. You do not need to set large numbers of breakpoints once you understand control flow principles. Breakpoints can then be used strategically to identify issues with specific variables or statements as they occur. If your program is not returning the output values you expect, or the program throws errors and you aren't sure why the errors are occurring, breakpoints can help.

### **Breakpoints with Jupyter Notebook**

Jupyter Notebook has the ability to set breakpoints, but the method is simplistic and a little tedious. The next section will introduce breakpoints in Jupyter Lab, which has breakpoint functionality that better mimics advanced IDEs. Consider using Jupyter Lab for debugging. Although the breakpoint feature in Jupyter Notebook is not as robust as in other IDEs, breakpoints can still be used to debug your code. To use breakpoints in your Python code with Jupyter Notebook, you must import the `set_trace()` method from the `IPython.core.debugger` library. To do this, you should add the following line of code to the first coding cell within your notebook:

```
from IPython.core.debugger import set_trace
```

Import statements work the same in Python as they do in other languages. An ***import statement*** specifies which built-in or external library of code you wish to use. The import statement above begins by telling the program which library to pull in (i.e., from `IPython.core.debugger`). Then, the specific method is identified (i.e., `import set_trace`) to avoid loading the entire `IPython.core.debugger` library. This practice saves computing resources.

To set a breakpoint, you would call the `set_trace()` method anywhere in your code that you want the program to stop when it executes. Write as many `set_trace()` methods as you need to identify the issue you may be experiencing. The program will stop at each `set_trace()` method as a breakpoint in the program. To check variable values at each breakpoint, you must use a simple console built into Jupyter Notebook. When you run a cell that has a `set_trace()` method in it, the cell output will contain a ***textbox labeled ipdb>*** and will stop at the first instance of `set_trace()` in the cell. Inside of the `ipdb` textbook, type the name of the variable you wish to examine and press Enter. Above the `ipdb` textbook, you will see the value of the variable or a `NameError` if the variable hasn't been initialized yet. You can enter another variable name and check that as well using the same method. Figure 3.1 shows the `ipdb` textbox in Jupyter Notebook.

**Figure 3.1: IPDB Textbox for Breakpoint Debugging in Jupyter Notebook**

```
11         raise NotAppleTextFileException()
12
13
14
15     set_trace()
16     manager = FileManager()
17     set_trace()
18     manager.readFile4("/home/apple/jeff.txt")
<ipython-input-1-12e6ef405827>(15)<module>()
13
14
--> 15     set_trace()
16     manager = FileManager()
17     set_trace()

ipdb>
```

After you have checked all of the variable values that you wish to analyze at that particular breakpoint, type the letter “c” (i.e., continue) into the ipdb> textbox and press Enter. **The “c” command** will tell the debugger to move to the next set\_trace() method in your code. You can then ask for variable values at the new breakpoint by typing the name of the variables in the textbox as described previously. If you wish **to terminate the breakpoint debugger**, type the letter “q” (for quit) into the ipdb> textbox and press Enter. You can start the breakpoint debugger again by clicking the “Run” button in the cell. If the debugger stops working, you can restart your notebook. After restarting the kernel, you can run the cell with the set\_trace() methods again.

### Breakpoints in Jupyter Notebook for Debugging

The code below shows the use of set\_trace() methods to create breakpoints in your Python code. Remember to import the set\_trace method as shown below.

```
from IPython.core.debugger import set_trace

class NotAppleTextFileException(Exception):

    def __init__(self, message="You must call the apple.txt file for the program to work."):
        Exception.__init__(self, message)

class FileManager:

    def readFile(self, fileLocation):
        set_trace() #the 3rd breakpoint
        if(fileLocation != "/home/jeff/apple.txt"):
```

```

set_trace() #the 4th breakpoint

raise NotAppleTextFileException()

#the code below is not part of a class. This is the starting point of the program
set_trace() #The 1st breakpoint (i.e., the program starts here)

manager = FileManager()

set_trace() #the 2nd breakpoint

manager.readFile4("/home/apple/jeff.txt")

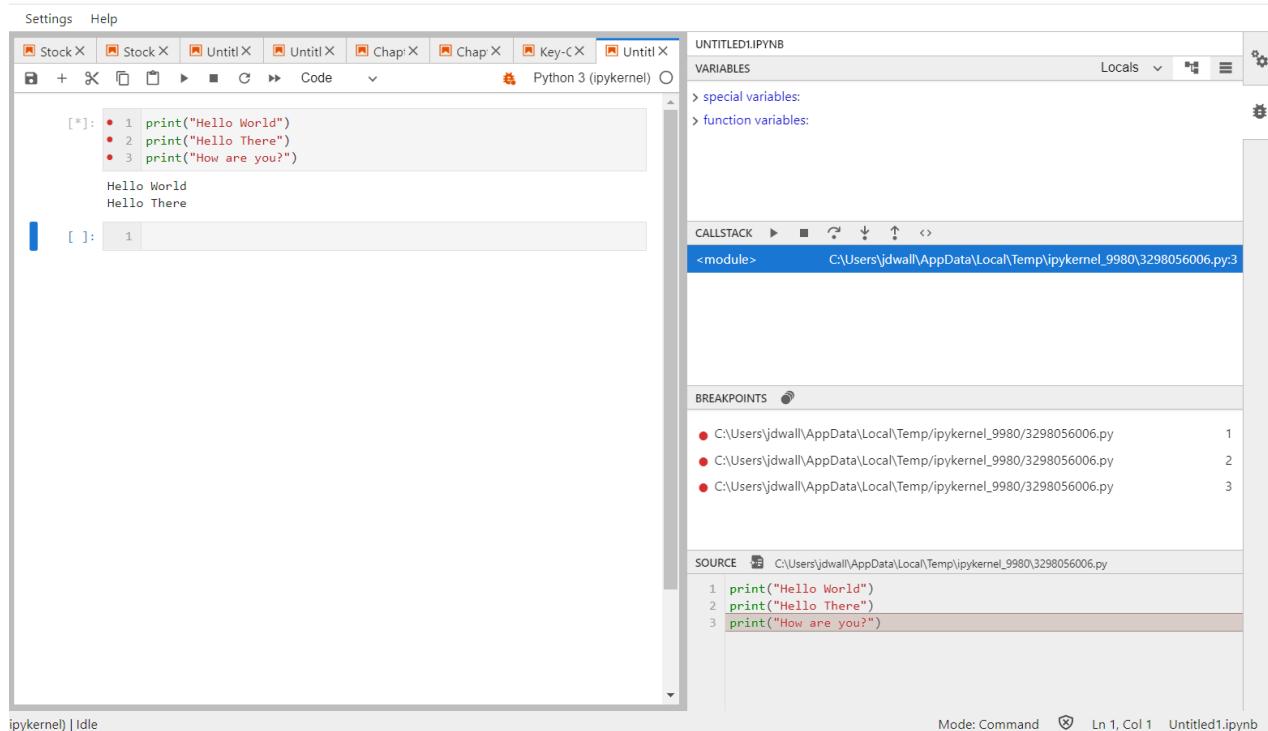
set_trace() #the 5th breakpoint

```

## **Breakpoints with Jupyter Lab**

Using breakpoints in Jupyter Lab is much simpler and also more similar to other advanced IDE's. Given the better debugging tools offered by Jupyter Lab, you should probably use Jupyter Lab for the assignments in the associated workbook. To use breakpoints in Jupyter Lab, you will want to click the debug icon mentioned in Chapter 1 on the right side of the interface. Clicking the icon will open the debugging window. Figure 3.2 shows the debugging window in Jupyter Lab.

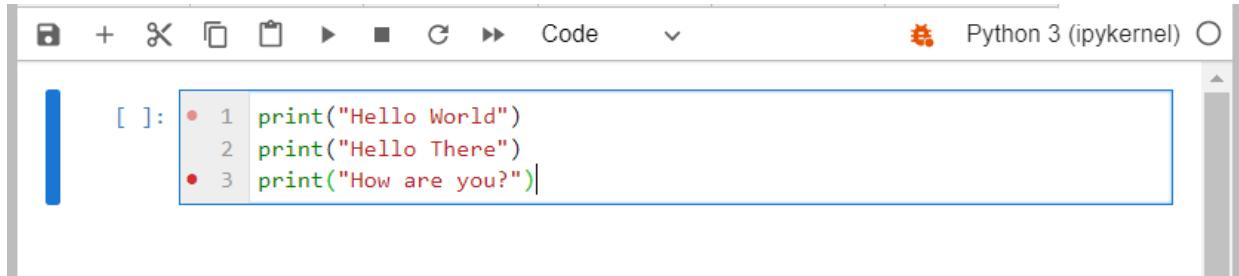
**Figure 3.2: The Debugging Window in Jupyter Lab**



The Variables section of the debugging window will allow you to see the values of variables at each breakpoint. This window will allow you to identify when a variable takes a strange value. The Callstack section will show you which command is being executed as you move through breakpoints. The play icon or the next icon in this section will move you from breakpoint to breakpoint. As you click the play or next icons, you will see the information in the Variables and Source sections change. The stop icon in this section will terminate the breakpoint session. The Breakpoints section simply shows the list of breakpoints that you can move through. The Source section shows you where in the code you are as you move through the breakpoints. Pay attention to this section to learn how programs move from line to line and method to method.

To set a breakpoint in your code, you will need to set the main coding window to debug mode. Do this by clicking the small beetle icon in the top right section of the coding window. The icon should turn orange when the debug mode is activated. Click it again to deactivate debug mode. You will see the small beetle icon turn a dark gray color. The debug mode will show line numbers next to each line of code in the cell. If you hover over a line number, you will see an orange dot appear next to the number. If you click the line number, the orange dot will change into a red dot that remains on the line. This red dot signifies that a breakpoint has been set for that particular line of code. You can add as many breakpoints as you would like to different lines of code in a cell or across cells. You don't need to set breakpoints on class and method declarations. You will primarily set breakpoints on lines of code within methods or on lines of code outside of classes and methods. Figure 3.3 shows the debug mode icon and the orange and red dots next to the code line numbers where you set breakpoints.

**Figure 3.3: Setting Breakpoints in the Code Window**



A screenshot of a Jupyter Notebook cell. The cell identifier '[ ]:' is on the left. Inside the cell, there are three lines of Python code: 'print("Hello World")', 'print("Hello There")', and 'print("How are you?")'. The first line has an orange dot at the start of the line, indicating a breakpoint. The second line has a red dot at the start of the line, also indicating a breakpoint. The third line has no dot. The top bar shows various icons for file operations, and the status bar indicates 'Python 3 (ipykernel)'.

Once you set your breakpoints as described above, you can use the play/stop/next icons in the Callstack section of the debugging window to move through the various breakpoints. For complex notebooks, it is often advisable to run all of your cells while not in debug mode to store variable information in memory. Then, you can run the debug mode with breakpoints. This practice is only necessary for notebooks with multiple cells that rely on classes in other cells to execute code. When running the debug mode in complex notebooks, you will want to run the Callstack control on the cell containing the starting point of your program and not on cells containing your classes.

**To start the entire debug process, you must choose a cell and click the run button in the main code window.** After clicking the play icon in the main code window, you can then use the Callstack controls to move through breakpoints. You will find the Jupyter Lab breakpoint controls much simpler than the coded method required in Jupyter Notebook.

## Using `print()` to debug

The `print()` method in Python was introduced previously because it is used regularly, including for debugging. The ***print() method can be used in a similar fashion to breakpoints*** in Jupyter Notebook. Some developers who learned to program before breakpoints existed prefer this method over breakpoints. You may enjoy it as well. The print debugging method uses `print()` statements to represent breakpoints and to print out variable values at specific points in the program. This approach offers similar functionality made available through the `ipdb` feature in Jupyter Notebook.

Consider that your program is breaking down but you aren't sure where. With `print()`, you can simply print out the line number or lines of code to find out where the system stops printing. This approach will tell you where the system breaks down. The system breaks down before the line number that is never printed.

Similarly, assume that you are getting the wrong value from a method or variable. You can print out variable values at specific points in the flow of control to find out when the variables are taking on wrong values. The code below shows these two uses of `print` methods for debugging. For example, the code below prints line numbers to the Console, such as on lines 3, 5, 7, 10, 12, 14, and 16. Assume after running the code below that the console only displayed the following:

```
line 3 of Program start()  
line 5 of Program start()
```

This console output would tell you that the code on line 4 is running correctly, but the program is breaking down on line 6. You know this because the `print()` method on line 7 is never executed. If it had been executed, you would have seen "line 7 of Program start()" printed to the console.

If a variable is changed several times during the program, you can print out the value to the console in multiple locations. This practice can help you identify where in the flow of control a variable is changing to an unexpected value. On lines 8 and 17 in the code below, the value of a variable called "age" is printed to the console. Assuming you fixed the error on line 6, you could check the values in the console and compare them to the values you expect to receive at each point in the flow of control. Printing a variable in multiple locations can help you diagnose when an incorrect value is introduced to a program.

	<b>Debugging with the <code>print()</code> method</b>
1	<code>class Program:</code>
2	<code>def start(self):</code>
3	<code>print("line 3 of Program start()")</code>
4	<code>customerName = "Jeff"</code>
5	<code>print("line 5 of Program start()")</code>
6	<code>age = 6/0</code>

```

7      print("line 7 of Program start()")

8      print("Age = {}".format(age))

9      mathObject = MyMath()

10     print("line 10 of Program start()")

11     mathObject.loadDataFromFile("C:/myfile/wrongFile.txt")

12     print("line 12 of Program start()")

13     mathObject.divide(6.0, 5.0)

14     print("line 14 of Program start()")

15     age = 45

16     print("line 16 of Program start()")

17     print("Age = {}".format(age))

```

If you use the `print()` method for debugging, you have to go back through your code and remove all `print()` statements after you fix any issues. The same is true for the `set_trace()` methods in Jupyter Notebook. This can be tedious. So you may prefer more advanced IDEs and true breakpoints like those in Jupyter Lab.

Whether you use breakpoints or `print` methods, be prepared to debug your code. You will run into errors at some point. Remember that if you can't figure something out, you should Google Exception or Error messages and read some programming forums. Not every "bug" in your code will have a fix that is obvious to you. You may need to do some reading to figure things out. You can even post comments to forums if you can't find an answer. You can post your code and ask for help. There are often many friendly developers who will answer your questions.

## AI Code Assistance

With advances in AI, namely large language models like ChatGPT, you don't always need to utilize the debugging techniques mentioned above. Some large language models can take code with errors and provide you with corrected code. Although these systems don't always function correctly, they can be a useful starting point for debugging. You can also write large language prompts to generate code, helping you minimize the tedium of programming. This might be particularly helpful if you are a citizen developer that likes creating new systems, but doesn't like writing code. If you are reading this text for a college course, please be sure to talk with your professor about whether you should use AI code assistance on assignments. Although AI code assistance is increasingly used in industry by developers, your professor may prefer that you learn appropriate coding techniques without assistance before adding assistance later.

## **Code assistance for debugging**

If you encounter an error while programming, tell a large language model that you are receiving an error message and then paste your code into the large language model prompt. The model will then provide suggested edits to the code to fix the code. It is best to provide the error message or error type and the programming language in the large language model prompt. For example, a prompt to a large language model might read:

I am experiencing an error in Python. I am using Jupyter Lab. Please provide me with corrected Python code to fix the error and a description of why the error occurred. I am getting the following error message {past error message here} for the following code {paste code here}. If you see any other errors, please fix those as well.

You will notice in the prompt above that the following features are included: the specific programming language (i.e., Python), the IDE used (i.e., Jupyter Lab), the particular error message received from the IDE, and the erroneous code are included. Although not all of this information is absolutely necessary, it often helps to be more specific when developing prompts for large language models. You will also notice in the prompt a request for corrected code. The large language model should provide you with updated code based on its understanding of coding and the error produced. In addition, the prompt requests an explanation of the error. Asking for explanations can help you learn about errors to avoid them in the future. Debugging can be very time consuming. If a code assist tool can help you or the developers you manage to debug, then be sure to utilize it.

If you are having a hard time understanding code, you can also **write a simple prompt to ask the code assist tool to explain the code and its functions**. The prompt would need to include a copy of the code you are trying to understand.

## **Code assistance for generating code**

In addition to debugging, some AI code assistance tools will also write code for you based on a textual description of what you want the code to do. Although this sounds amazing, the technology is still in its infancy. It does not always produce error-free code. It can produce insecure code at times, and it will not write you an entire software system with a single prompt.

Current AI code assistance technology requires that you possess a good understanding of programming concepts, such as objects, functions, etc. To use these tools for programming, you have to be able to break down a larger program or system into its smallest components (i.e., functions/methods). Once you conceptually break down the system to these components, you can develop prompts to build each of the components.

For example, if you wanted to build a financial calculator to calculate a specific set of financial ratios, you would need to make some decisions before starting. For example, you would need to decide if you were going to rely solely on procedural programming or use object-oriented programming. These decisions will determine how you write the prompt. Let's assume you want to create the calculator in a class named FinancialRatiosCalculator. Next, you would need to determine whether you wanted the ability to ask for specific financial ratios, or simply generate the full list of financial ratios every time. Let's assume we want the precision to select specific ratios. We would then need a separate function to calculate each ratio. You

must also decide how you want to display the results. Is the function simply going to return a value that will then be displayed by some other code, or do you want to use a simple print() function to display the data. For simplicity, we will use a simple print() function. With this information, you can begin to develop a prompt.

If you want your program to follow a specific paradigm, specificity is key when writing large language model prompts. For example, you could write a simple prompt, such as:

Write Python code to create a financial calculator to calculate common financial ratios

The large language model will provide a code output, but in this example, Google's Bard produced procedural programming with a random subset of ratio functions. This might not meet the specific requirements of the calculator you want.

The following prompt could be written to generate code with a little more specificity:

Write Python code that will run in Jupyter Lab that includes a class named FinancialRatiosCalculator. The class should include the following functions: calculatePERatio() that should calculate the PE ratio of a stock and print the result with a label using the print() function, calculateDebtToEquityRatio() that should calculate the debt to equity ratio for a stock and print the result with a label using the print() function.

Although this prompt provides some detail, it isn't as detailed as we might want it. For example, Google's Bard language model produced the following output for this prompt:

```
class FinancialRatiosCalculator:
```

```
    def __init__(self, price, earnings, debt, equity):
        self.price = price
        self.earnings = earnings
        self.debt = debt
        self.equity = equity

    def calculate_pe_ratio(self):
        return self.price / self.earnings

    def calculate_debt_to_equity_ratio(self):
        return self.debt / self.equity

if __name__ == "__main__":
    calculator = FinancialRatiosCalculator(100, 10, 50, 50)
    print("PE Ratio:", calculator.calculate_pe_ratio())
    print("Debt to Equity Ratio:", calculator.calculate_debt_to_equity_ratio())
```

In the code above, the language model made the decision to require all inputs for the calculations (i.e., price, earnings, debt, and equity) to be contained in the constructor (i.e., \_\_init\_\_()) method. This doesn't really help with our interest in being able to call specific ratio functions individually. Similarly, the print() methods aren't contained in the functions, but are instead included in the \_\_main\_\_ function that runs all

of the ratios when the program is run. This also doesn't provide the freedom to print individual ratios from our calculator. To fix this, we can write a more specific prompt, or write a series of related prompts. The next example shows how to chain prompts together to create code.

First prompt:

Write Python code that will run in Jupyter Lab that includes a class named FinancialRatiosCalculator. Include the constructor method, but nothing else.

First result:

```
class FinancialRatiosCalculator:
```

```
    def __init__(self):  
        pass
```

Second prompt:

Within the FinancialRatiosCalculator, add a method named calculatePERatio(). The calculatePERatio() should accept a price and earnings for a stock and calculate the ratio. The calculatePERatio() method should also print a message for the user stating what the PE ratio is for the stock.

Second result:

```
class FinancialRatiosCalculator:
```

```
    def __init__(self):  
        pass
```

```
    def calculatePERatio(self, price, earnings):
```

```
        """
```

Calculates the price-to-earnings ratio for a stock.

Args:

price: The price of the stock.

earnings: The earnings per share of the stock.

Returns:

The price-to-earnings ratio.

```
        """
```

```
        pe_ratio = price / earnings
```

```
        print(f"The PE ratio for the stock is {pe_ratio}")
```

```
        return pe_ratio
```

Additional prompts could be written to include other specified ratios. Further, prompts can be added to make changes to methods you have already created. For example, if you wanted to store the PE ratio

value from the method in the example above within a class variable (i.e., an instance variable), you could writing something like:

Add an instance variable to the FinancialRatiosCalculator named peRatio and store the result produced by the calculatePERatio() method in the instance variable. Show the full code.

Such results might change other aspects of the code that you wanted, so be sure to check the results carefully. For example, the prompt above removed the print() function from the calculatePERatio() method.

Result:

class FinancialRatiosCalculator:

```
def __init__(self):
    self.pe_ratio = None

def calculatePERatio(self, price, earnings):
    """
    Calculates the PE ratio for a stock.

```

Args:

price: The price of the stock.

earnings: The earnings per share of the stock.

Returns:

The PE ratio.

"""

```
    pe_ratio = price / earnings
    self.pe_ratio = pe_ratio
    return pe_ratio

```

You could simply prompt the tool to add the print() statement back to the method. The point here is that specificity is crucial and that the tools can be unpredictable. Be sure to check the output after each prompt to make sure that you haven't lost important information. AI code assistance tools are in a nascent state. Although they can be helpful, they can also be frustrating. Don't expect too much from these tools at this current time. These tools will continue to develop and improve over time.

## Testing Software

When you are working for a large organization and many people are writing code for a large system, how do you ensure that a poor code change made by one software developer does not affect the entire system? In many programming languages, a single error can cause an entire system to fail if precautions are not taken.

A big part of the answer to this question is software testing. You should always test your code before "going live" (i.e., making your program available to users on production servers). There are many types

and levels of testing. **Unit tests** are the most granular tests. Unit tests ensure that small parts of your system, usually individual methods, work as expected. For example, you might want to test that a divide() method actually produces the output you expect for several meaningful inputs (e.g.,  $1/5 = 0.2$ ,  $2/2 = 1$ ,  $0/9 = 0$ ,  $123456789/4898465 = 252.2198$ , and  $9/0$  throws an Exception). This text will focus on unit testing. However, other types of testing are also important to ensure you produce consistently quality software applications.

Once you know that individual methods work correctly, you will want to make sure that interactions work properly between classes and methods in OOP or between functions in procedural programming. This type of testing is called **integration testing**. Integration testing focuses on modules of functionality, usually derived from multiple methods/functions and/or classes working together. The next step in testing is **system testing**, which tests whether all of the modules function well together as an entire system. There are many other types of tests as well, such as acceptance testing, in which a set of users test your software to determine if it has the functionality they desire; stress testing; and more. Careful testing is an integral part of software development. A quality IT department will engage in many testing efforts to ensure the quality of the IT products and services they provide.

Traditionally, the task of testing was accomplished by many quality assurance (QA) personnel. Developers would write code and commit code to a central repository (i.e., a storage location for all of the code developers write for a system). Then, QA teams would run the code in a testing environment to see if the program functioned as expected. **Quality assurance**, ensuring code is bug free and functions as expected, is crucial for good software development practice in organizations. Users and customers expect working and high-quality software. The problem with quality assurance under the traditional model was and is the costs associated with manually testing software. Staffing large teams of manual testers increases the cost of software development, and humans don't always catch every error.

How can organizations reduce testing costs and still develop quality software? Part of the answer is automated testing. **Automated testing** is the practice of writing code (e.g., unit tests) to test your software and utilizing automation software to run the various tests. Many languages possess built-in testing "languages" to help you test your code. In Java, JUnit is one of the standard libraries for unit testing. In Python, unittest and many other libraries exist to support unit testing. Although automated testing is more widely used than ever, it should be noted that automated testing is not a panacea. Humans still have to write the automated tests and some errors may still be missed. Automated testing is only as useful as the processes and practices that support it. Poorly written automated tests are not better than quality manual tests.

## **Unit testing in Python with unittest**

Python has several unit testing libraries available to you. In this chapter, we introduce the unittest library. To use the unittest library, you first need to import it into your program. Within a notebook, this can be accomplished by adding the following line of code to a cell (usually the first cell) in the notebook:

```
import unittest
```

With the unittest library imported, you can write test cases for methods in your Python programs. For the complete documentation of how to use the unittest library, visit:

<https://docs.python.org/3/library/unittest.html>.

Unittest is used to test classes and methods/functions within your program. The example below represents a class with methods that you might want to test. The divide method could be tested for various uses, such as a zero value in the numerator returns a value of zero, a zero value in the denominator raises a ZeroDivisionError(), a division of two numbers with a non-repeating value returns a non-repeating decimal (e.g., 1/4 returns 0.25), and division of two numbers with a repeating value returns a repeating decimal (e.g., 1/7 returns 0.142...).

#### Example of a Class to be Tested

```
class SimpleMath:  
    def divide(self, numerator, denominator):  
        if denominator == 0:  
            raise ZeroDivisionError()  
  
        return numerator / denominator
```

Unittest test cases in Python are contained within classes. The test case classes that you create must inherit from the unittest TestCase class to perform the behaviors of a test case. Unittest operates based on assertions represented by assert methods. An **assertion** tests whether a condition is true or false.

The code below demonstrates how to create a new unittest test case in Python. Notice in the code below that after creating the SimpleMathTest class, which inherits behavior from “unittest.TestCase”, there is an if statement. The if statement is what causes the test case to run. For Python programs created in Jupyter Notebook, a slightly different syntax is required to test the notebook.

#### Example of a Test Case with unittest

```
import unittest  
  
class SimpleMathTest(unittest.TestCase):  
    def testThatZeroNumeratorReturnsZero(self):  
        math = SimpleMath() #instantiate the class to be tested  
        self.assertEqual(math.divide(0,15), 0) #assert the method values are as expected  
  
#for generic Python programs that are not run inside of a Jupyter Notebook use this  
if __name__ == '__main__':  
    unittest.main()  
  
#for Python programs that are run inside of a Jupyter Notebook use this  
if __name__ == '__main__':  
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

## unittest assert methods

Unittest contains a series of different assert methods. A list of more of the assert methods can be found at the following link: <https://docs.python.org/3/library/unittest.html#classes-and-functions>. Some primary assert methods to consider are provided in the table below.

Assertion Type	Description	Examples
<code>assertEqual(actualValue, expectedValue)</code>  For doubles/floats: <code>assertAlmostEqual(actualValue, expectedValue, numberOfDecimalPoints)</code>	The assertEquals() method tests whether the result of a method returns the expected value of the method given a specific set of inputs. If the values are equal, the test passes. If the values are not equal, the test fails.  If you are using floats or doubles, the assertAlmostEqual() method should be used. This method requires that you provide a precisionValue (i.e., the number of decimal points to look at when comparing two values).	#this is the method to test <code>class MyMath:</code> <code>def multiply(self, n1, n2):</code> <code>    return n1 * n2</code>  #test methods would be in the test class #this test would pass <code>def testOfEqualityExample1(self):</code> <code>    math = MyMath()</code> <code>    self.assertEqual(10, math.multiply(5,2))</code>  #this test would fail <code>def testOfEqualityExample2(self):</code> <code>    math = MyMath()</code> <code>    self.assertEqual(11, math.multiply(5,2))</code>  <code>def testForFloatsAndDoubles(self):</code> <code>    math = MyMath()</code> <code>    self.assertAlmostEqual(0.143,</code> <code>        math.divide(1,</code> <code>            7), 3)</code>
<code>assertFalse(testCondition)</code>	The assertFalse() method tests whether the result of a method returns a boolean false value. If the return value is false, the test will pass. If the return value is true, the test will fail.	#this is the method to test <code>class Person:</code> <code>def isTall(self, height):</code> <code>    if height &gt; 6:</code> <code>        return True</code> <code>    else:</code> <code>        return False</code>  #test methods would be in the test class #this test would pass <code>def testOfFalseExample1(self):</code> <code>    p = Person()</code> <code>    self.assertFalse(p.isTall(4))</code>  #this test would fail <code>def testOfFalseExample2(self):</code> <code>    p = Person()</code> <code>    self.assertFalse(p.isTall(8))</code>
<code>assertTrue(testCondition)</code>	The assertTrue()	#this is the method to test

	<p>method tests whether the result of a method returns a boolean true value. If the return value is true, the test will pass. If the return value is false, the test will fail.</p>	<pre>class Person:     def isTall(self, height):         if height &gt; 6:             return True         else:             return False  #test methods would be in the test class #this test would pass def testOfTrueExample1(self):     p = Person()     self.assertTrue(p.isTall(8))  #this test would fail def testOfTrueExample2(self):     p = Person()     self.assertTrue(p.isTall(4))</pre>
assertIsNone(testCondition)	<p>The assertIsNone() method will test whether a value that should be None is actually returned as None. If the return value is None, the test passes. If the return value is anything but None, the test fails.</p>	<pre>#this is the method to test class APICaller:     def getResponse(self, json):         if json.content == None:             return None         else:             return json.toString()  #this test would pass def testOfNoneExample1(self):     json = JSON()     caller = APICaller()     self.assertIsNone(caller.getResponse(json))  #this test would fail def testOfNoneExample2(self):     json = JSON()     json.setContent("{age: 49}")     APICaller caller = new APICaller()     self.assertIsNone(caller.getResponse(json))</pre>
assertRaises(NameOfError)	<p>Checks to see if a specific Error is raised when it should be thrown. If the specified error is thrown, the test passes. If the specified error is not thrown, the test fails.</p>	<pre>#the class to test class MyMath:     def divide(self, num, den):         if den == 0:             raise ZeroDivisionError()  #this test would pass def testDivByZeroThrowsException1(self):     math = MyMath()      with self.assertRaises(ZeroDivisionError):         math.divide(7,0)  #this test would fail</pre>

		<pre>def testDivByZeroThrowsException1(self):     math = MyMath()      with self.assertRaises(ZeroDivisionError):         math.divide(0,7)</pre>
--	--	--

With the methods in the table above, you can write a variety of different test cases. To run the tests, make sure that the if statement in the prior code example is included at the end of the test case. Then, just run the cells in the notebook and it will run the tests for you. If a failure occurs, an error message will be displayed for each failed test. It will also tell you how many failed tests occurred. If all tests succeeded, the results will show how many tests were run and then display: OK.

When writing your tests, **do your best to avoid turning unit tests into integration tests**. Often, through inheritance or composition, two or more classes can be interlinked. This interconnection between classes can make it difficult to test individual methods in isolation. However, where possible, you should avoid instantiating more classes than the one you are testing. When data or other classes are necessary, you can use mocks or stubs to mimic the structure of data or the functionality of related classes. We will not explore mocks and stubs in this text. However, you can read more about them by searching topics like “Python unittest mocks.”

## Pair Programming

Because of the exponential growth of information technology and information systems, there is a constant need for people who know how to write and maintain programs. What this also means is that as long as this exponential growth continues, there will be a high percentage of entry-level/junior developers in industry; sometimes as high as 50% of developers in industry have less than 5 years of experience. This constant need for new developers creates training issues for organizations. How do you take the rudimentary programming skills that junior developers possess and train them to become senior developers?

Some organizations hold weekly, bi-weekly, or monthly programming training presentations. A senior developer or a budding junior developer teaches a programming concept during these meetings. Other organizations are complementing training with a practice called pair programming. Pair programming is a practice that is a part of the extreme programming (XP) discipline. You will learn more about XP in courses and texts about systems analysis and design. Feel free to Google the term if you are interested in learning more about XP.

**Pair programming** is the practice of assigning two programmers to a single programming task. Although there are different versions of pair programming, the original practice calls for both developers to use only one computer, one keyboard, and one mouse. One of the members of the pair writes code, while the second member of the pair reads the code as it is written and makes suggestions to improve the design, or asks why a particular design choice was made. The pair then switches roles throughout the day so that one person is not always programming or observing/commenting.

Pair programming is meant to reduce the number of bugs in code and increase the quality of program design by keeping two pairs of eyes on the code at all times. Although this may seem inefficient, after you realize how long it can take to debug a program or to improve code, you will recognize the value of getting

things mostly right the first time. However, pair programming is not a silver bullet and bugs still leak into programs even with two minds working on the same code.

In practice, programming pairs often consist of a senior member and a junior member. ***This practice is used to help train junior developers.*** The junior developers are allowed to ask many questions, see how senior developers write code, and write code under the guidance of an expert. The practice can feel strange at first, but many learn to love it.

Other variants of the pair programming have each person in the pair working on the same programming task, but with two computers and different subtasks. One such variant has one member of the pair writing the code for the program and the other member writing automated unit tests to validate the quality of the code.

Ask your professor if you can work on programming assignments in pairs. As long as you are switching off between who is writing and who is observing/commenting, it can be a valuable tool to learn programming. Remember that pair programming is designed for face-to-face interaction. Simply sharing files via email or in other remote fashions does not give you the same level of mentorship. Use the method to help you learn, instead of using it to allow others to do your work. Pair programming is a learning tool and a quality improvement tool. Pair programming is a collaboration practice and NOT a delegation practice.

## Chapter 3 Assignment

Please see the [Chapter Assignments Workbook](#) for the Chapter assignment.

## Section II: Data in Business Programming

This section of the text is dedicated to explaining how you can use business data within computer programs to automate business operations or assist in business decision making. Section II includes Chapters 4, 5, 6, and 7.

**Chapter 4** will introduce you to the idea of data structures, which are simply different ways to store data in computer memory. You will learn how to use basic data structures like arrays, lists, and dictionaries that are useful in business systems and for business analytics calculations. You will also be introduced to more complex data structures (i.e., tree and graph data structures), but from a conceptual perspective only. Programming these advanced structures is beyond the scope of this text. However, some of these concepts are brought up in later sections.

**Chapter 5** will introduce you to how data structures can be used in programming logic to automate business decisions. This section will introduce conditional logic to help you automate the processing of business options and alternatives. You will also learn how to loop over data structures to repeat business logic on data. Different types of loops are introduced, such as for and foreach loops, as well as recursion. After completing this chapter, you will have a basic understanding about how programs use data to automate business processes. You will not be an expert in automation yet, but will possess a foundation to build on.

**Chapter 6** will introduce you to different tools and techniques for conducting business analytics. Python is a programming language that is used heavily for business analytics purposes. Important Python libraries for analytics are introduced in this chapter. Using these libraries, you will learn some simple forms of data wrangling (i.e., cleaning data sets for statistical analysis). You will also be exposed to basic ideas from descriptive statistics. You will find that with a few simple programming commands, you can calculate statistics that are tedious and sometimes difficult to calculate manually. Chapter 6 also describes how to create some simple data visualizations, such as scatter plots and time series plots. By the end of this chapter, you will possess enough knowledge to explore simple business data sets.

**Chapter 7** will introduce you to database technologies and how data from databases can be integrated into a computer program. Businesses rely heavily on databases to store important information about customers, marketing campaigns, capital and other assets, supply chain partners, inventory and more. Understanding what databases are used for and how to access data within them will further your ability to conduct business analytics inquiries to make better management decisions.

After completing this section, you will possess foundational information about how data is stored and used within business information systems to support the needs of business users. Your knowledge will still be introductory, but the foundational information from these chapters will support further exploration of business analytics topics. The assignments in the associated workbook will help you to practice working with data structures, databases, and business analytics techniques.

# Chapter 4:

## Introduction to Data Structures

In Chapter 2, you were introduced to the concept of variables and primitive data types. You were also introduced to classes and objects, which are more complex data types, or more appropriately, data structures. In this chapter, we will expand upon the concept of data structures by introducing common data structures used in many types of programs. In the next chapter, you will learn how to integrate data structures into the flow control of your programs to create more complex functionality.

Simply put, a **data structure** provides a way to store, organize, access, and modify data in computer memory. Data structures are used in place of and in conjunction with primitive data types. In fact, classes and objects with their attributes represent a type of data structure. For example, a Customer class with firstName, lastName, and phoneNumber attributes represents a custom data structure to store, organize, access, and modify data about a customer. The methods of a class are often used to modify or display the data structure elements within the class.

You have been introduced to classes and attributes, but we haven't discussed other types of common data structures. This chapter will introduce **arrays**, **lists**, and **dictionaries** in Python. There are many other data structures, such as stacks, queues, trees, and graphs. A full coverage of different data structures isn't possible in this text. There are entire texts and courses dedicated to understanding and creating data structures. Learning about arrays, lists, and dictionaries will give you a foundation in basic data structures to help you write the types of programs used in many business settings. As you learn to use arrays, lists, and dictionaries, you will feel more comfortable reading resources about more complex data structures like trees and graphs. This chapter also conceptually introduces trees and graphs without showing how to create or use them with code.

### Foundational Data Structures

Different data structures possess certain qualities, such as the speed of accessing data, the speed of inserting data, the size of memory allocated for the data structure, and the manner in which data can be inserted and accessed. For example, **stack data** structures are designed to support processes that rely on last-in-first-out (LIFO) structures. Think of LIFO as a stack of paper upon which you add new paper to the top and always take paper from the top as well. Thus, the last paper placed on top of the pile will be the first paper taken off of the pile. **Queues** on the other hand rely on first-in-first-out (FIFO) structures. Think of queues as a line of people at the grocery store checkout. The first person to arrive in line at the checkout will be the first to be processed and the first one to leave the line.

LIFO and FIFO models are used in business, such as in accounting to manage inventory costs (e.g., do you consider the cost of a good sold as the sale of the first inventory item purchased (FIFO) or the most recent inventory item purchased (LIFO)). Watch for naturally occurring data structures within business operations. Doing so will help you to identify the best data structures to use in business programs. Let's explore some basic data structures that will help you write more complex programs in Python.

## Arrays

An **array** is an ordered collection of items in which the items are represented sequentially by integers called indices. For a simple example, consider a numbered grocery list. The numbers of the list would represent the indices of grocery items. If I asked what the item at index #1 was, you could easily respond: 6 bananas. Arrays operate in a similar fashion. Information is stored at each index and the value stored at that index can be accessed by referencing the index.

1. 6 bananas
2. 4 apples
3. 1 bag of lettuce

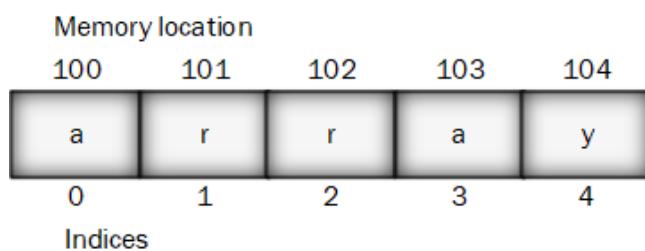
You can quickly iterate through the items in an array through the use of the indices. **Iteration** means moving sequentially from one item to the next in a collection. In many programming languages, including Python, the item indices in an array start with a **zero index**. That means you start iterating over the sequence at item 0 instead of at item 1 like you would when counting. So, the first item in the array has an index of 0, the second has an index of 1, and so on. This is important to remember, because if you want the 3rd item in an array, you must ask for the 2 index. Simply subtract 1 from the  $i^{\text{th}}$  item you wish to retrieve and that will be the index you need to reference. In programming, array indices are often surrounded by hard brackets (e.g., [0], [1], etc.). So, the grocery list above in programming syntax would look more like this:

- [0] 6 bananas
- [1] 4 apples
- [2] 1 bag of lettuce

In programming languages, a true **array stores items contiguously in memory** (i.e., stored in the same "chunk" of computer memory). As such, arrays take up less space in computer memory than data structures that are not stored contiguously. Because array items are stored in one chunk of memory, data can also be retrieved very quickly from arrays as compared to other data structures. To ensure that the items in an array are stored contiguously in memory, the size of an array (i.e., how many items it can hold) is set when the array is first created. Once set, the size of a true fixed-size array does not change. We will learn that this is not always the case when we examine list data structures (i.e., dynamic arrays). For now, just assume that arrays must be of a fixed size. Figure 4.1 shows a conceptualization of an array consisting of five letters that together spell "array". The example shows fake memory addresses.

Importantly, you will notice that the memory addresses are in a sequence as they are stored contiguously (i.e., consecutively in one "chunk" of memory).

**Figure 4.1: Contiguous Storage of Items in an Array**



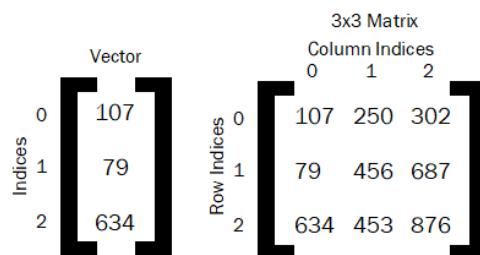
In some languages like Java, if you try to add extra items to a fixed size array, the program will throw an Exception, such as the ***ArrayIndexOutOfBoundsException*** in Java. This Exception simply means that you are trying to add more items to the array than you set it up to hold. Arrays can be rather rigid.

However, they minimize the use of memory and can retrieve items quickly because items are stored in contiguous blocks of memory. There are clear uses for arrays. In fact, you will see them used regularly in business analytics programs. If the size of the collection of items is known beforehand, is not too large, and your use case calls for fast retrieval of items by an index, then an array could make an excellent data structure.

Arrays are often used to create mathematical vectors and matrices in analytics programs. A vector is represented by a single array. Matrices are simply multidimensional arrays, meaning they are represented by storing arrays of information inside of an array (i.e., an array of arrays). Figure 4.2 shows a vector and a 3x3 matrix with array indices. Matrices present the row index first followed by the column index: `matrix[number of rows][number of columns]`. In Figure 4.2, the value at (0,2) or [0][2] (i.e., row 0 and column 2) is 302 and the value at (2,0) or [2][0] (i.e., row 2 and column 0) is 634. What is the value at (1,2) or [1][2]?

If you wish to become a data analytics professional, you should take some time to understand matrix-based mathematics, such as matrix multiplication, the transposition of matrices, and other related concepts. You can learn about matrix-related mathematics in linear algebra courses. Many analytics focused programming languages have built-in data structures called matrices so that you don't need to build a matrix from scratch.

**Figure 4.2 Arrays as Vectors and Matrices**



Python provides several built-in array implementations. However, most of them act more like lists than as true fixed-size arrays. Although the built-in Python library does not have a traditional fixed-size array, the ***numpy library***, a commonly used data analytics library for Python, has a fixed-size array. The fixed-size arrays in numpy are used in a number of machine learning and analytics libraries in Python. If you try to add more elements to a numpy array than it was set up to hold, the Python interpreter will throw an `IndexError` with a message that the "index [index number] is out of bounds."

The numpy library has many useful tools for analytics programming, some of which we will explore throughout this text. To use the numpy library, you must ensure that the numpy library is installed on your computer or server. If you installed Jupyter Notebook through Anaconda, numpy should come pre-installed. If you receive an error stating that numpy is not installed on your system, such as when trying to import it into your program, view the documentation on the numpy site: <https://numpy.org/> and install it on your computer. Examine the example below to see how you can use fixed-size arrays with the numpy library.

## Declaring, Instantiating, and Initializing Arrays/Vectors

When using numpy arrays, you must first import the numpy library. Often, professionals will add an **alias** to their imports, such as: **import numpy as np**. The alias "as np" in this example allows you to call the classes and methods of the numpy library with an abbreviation. So instead of calling `numpy.array()`, you could call `np.array()`. Aliases can be helpful for writing calls to imported libraries with long names. You will commonly see aliases used in programming forums and in practice.

Arrays in numpy are created from the **ndarray** class. You can either create an ndarray using the ndarray class directly, which is not commonly done, or by calling one of the specialized methods that create a specialized ndarray. Some of these specialized methods include the: **array()**, **zeros()**, **ones()**, and **empty()** methods.

The **array()** method can be used to create an ndarray from the dynamic array lists built into Python. In Python, built-in arrays are contained in hard brackets: [123, 456, 798, 231]. Simply add this hard bracketed list to the numpy `array()` method as an argument and your dynamic array will be converted to a numpy fixed-size array: `np.array([123, 456, 798, 231])`.

The **zeros(size of array,data type of array)** method creates an ndarray of the specified size and data type. Each item in the array is initialized with the value 0. The **ones(size of array,data type of array)** method creates an ndarray of the specified size and data type. Each item in the array is initialized with the value 1. Finally, the **empty(size of array,data type of array)** method creates an ndarray of the specified size and data type with random values at each index.

After creating an ndarray, you can alter the values at each index by calling to the specific index and setting the value, such as in lines 11-16 below.

You can access the value at a particular index by calling to the index: `print(ages[2])`.

```
1 import numpy as np
2
3 class ArrayExample:
4
5     def executeArrayExample(self):
6
7         #generally you will not directly create an ndarray like the line below
8
9         ages = np.ndarray(5,float) #array of floats of size 5
10
11        #instead you will create an ndarray through one of the methods below:
12
13        ages1 = np.zeros(3,int) #array of integers of size 3 filled with the value 0
14
15        ages2 = np.ones(5,float) #array of floats of size 5 filled with the value 1
16
17        ages3 = np.empty(5,float) #array of float of size 5 with random numbers
18
19        #each array item can be assigned new values like in the syntax below
```

```

11      ages[0] = 25
12      ages[1] = 23
13      ages[2] = 46
14      ages[3] = 62
15      ages[4] = 156
16      #you can also create an ndarray from a built in Python array
17      #built in Python arrays are contained within hard brackets []
18      salaries = np.array([22500.00, 50000.00, 95000.00]) #array of float of size 3
19      print(salaries[1]) #this would print 50000.00

```

The numpy library also allows for the creation of matrices (i.e., multidimensional arrays).

### **Declaring, Instantiating, and Initializing Matrices**

Matrices can be represented with ndarrays in Python by including two values for the array size contained within parentheses. The following code creates a matrix with the numpy library:

**employeeSalaryChange = np.zeros((10,11), float)**. The code would create a new ndarray with 10 rows and 11 columns (i.e., a 10x11 matrix) that would store float values initialized to zero at each index pairing.

The matrix in the example below is called weightChangeMatrix. The matrix could be used to store data about the weight change of objects over time. The rows in the matrix would represent four different real-world objects (e.g., pens, coins, computers, etc.). The columns would represent a measurement of the weight of each object at three time points. You can change the value of each row by column index pair as depicted below.

You can access the value at a particular row by column index pair by calling to the index pair:  
`print(ages[2][0]).`

```

import numpy as np

class MyMatrix:

    def matrixExample(self):
        #the matrix would have 4 rows (objects) and 3 columns (weight measurements)
        weightChangeMatrix = np.zeros((4,3),float) #a 4x3 matrix of floats

        weightChangeMatrix[0][0] = 107 #setting the value for row 0 column 0 to 107

```

```

weightChangeMatrix[0][1] = 250 #setting the value for row 0 column 1 to 250

weightChangeMatrix[0][2] = 302 #setting the value for row 0 column 2 to 302

weightChangeMatrix[1][0] = 79 #setting the value for row 1 column 0 to 79

weightChangeMatrix[1][1] = 456 #setting the value for row 1 column 1 to 456

weightChangeMatrix[1][2] = 687 #setting the value for row 1 column 2 to 687

#setting values for the other cells would continue for all 4 rows (i.e., rows 0-3)

#alternatively, you could create the same matrix with arrays embedded in another array
#notice that the outer array contains three inner arrays: [ [], [], [] ]

weightMatrixAlternate = np.array([ [107, 250, 302],
                                  [79, 456, 687],
                                  [634, 453, 876] ], float)

print(weightChangeMatrix[0][2]) #would print 302

```

You now possess enough knowledge to create and use vectors and matrices with numpy for analytics and machine learning purposes. This knowledge will be needed in later chapters as you learn to manipulate and use data analytics and machine learning techniques.

## *Lists*

Python also has list data structures. Like an array, **lists** are ordered collections of items organized by sequential indices. Again, lists tend to start with a 0 index. The main difference between lists and arrays is that lists can be resized, while true fixed-size arrays cannot. Technically, all arrays built into Python (except those in external libraries like numpy) are lists implemented as dynamic arrays (i.e., array lists). We use the terms array list and dynamic array interchangeably in this text. Dynamic arrays allow you to add new items to an array after initializing it.

The numpy ndarray only allows for primitive data types like integers and floats to be stored within. However, built in Python array lists accept all data types, including objects instantiated from classes. Python lists can also contain multiple different data types instead of one single data type. Array lists in Python are quite flexible. However, with flexibility comes the danger of creating unexpected errors due to incorrect data types.

In languages like Java, there are two ways to implement lists, namely the ArrayList (a.k.a., dynamic arrays) and the LinkedList. In Python, only array lists are built into the Python library. External libraries can be used to add linked lists to your Python code if you need a linked list.

An **array list** is similar to an array, except that the array list can be resized (i.e., a dynamic array). A list will maintain an unbroken sequence of indices by shifting indices when items are deleted or added. When you delete an item from a list, all of the indices for items after the deleted item will be decreased by 1. For

example, assume you had index:value pairs as follows: [0] = 10, [1] = 45, [2] = 56, [3] = 60. If you deleted index 1 (i.e., the value 45), the resulting index:value pairs would be shifted to be: [0] = 10, [1] = 56, [2] = 60. Index 2 was shifted to index 1 and index 3 was shifted to index 2.

## Array lists

Let's assume you have an array list that is storing three Customer objects. Behind the scenes, Python will store these three elements contiguously in memory in an array. If you want to add a 4th object to the array list, Python copies the three elements from the original array, creates a new larger array, and adds all four elements to the new array. Then, the original array that held the three items is deleted. All of this happens behind the scenes as you increase the number of items in the list. Of course, this is a bit of a simplification of what actually happens, but enough to help you understand how array lists work. Dynamic arrays are typically initialized with more items than you might actually need. When the array needs to be resized, the new array is also given extra size to accommodate growth in items. This practice is done to minimize the number of times a new array needs to be created to improve efficiency.

Because of this behind the scenes work, adding new elements to or removing elements from an array list can be a little more time consuming than other list implementations, such as linked lists, particularly when the number of items in the array is large. However, arrays and array lists can retrieve data faster than other types of lists, such as linked lists, because values are stored contiguously in memory. Often, this difference is negligible because of the speed of modern computers. The following code shows the use of array lists in Python.

### Declaring, Instantiating, and Initializing Dynamic Arrays as Lists

Python has several ways to declare a list as a dynamic array. In this section, we will explore the simplest and most common method to create a list. To create a list in Python you will write the list items in hard brackets and store them in a variable: `list = [45, 48, 65, 5]`. You can add elements to a list by calling the `append()` method on the list: `list.append(26)`. You can also **remove specific values** from a list: `list.remove(48)`. Further, you can **remove values by their index** by calling the `pop()` method and passing it the index of the item you wish to remove. For example, `list.pop(2)` would remove the value at index 2 from the list. The code below shows several lists with the add and remove methods.

You can access the value at a particular index by calling the index: `print(list[1])`.

For a list of all of the methods you can call on a Python array list, see:

[https://www.w3schools.com/python/python\\_arrays.asp](https://www.w3schools.com/python/python_arrays.asp).

```
class MyMath:  
  
    def executeListExample(self):  
  
        ages = [45, 16, 25, 34, 79] #list of integers of size 5  
  
        ages.append(67) #adds the value 67 to the end of the ages list (i.e., at index 5)  
  
        ages.remove(16) #removes the value 16 at index 1 from the list
```

```

ages.pop(0) #removes the value at index 0 from the list

salaries = [22500.00, 50000.00, 95000.00] #list of floats of size 3

multiDataTypeList = [12, "Utah", 47.6] #list of multiple data types of size 3

print(multiDataTypeList[1]) #would print Utah

```

**Array lists can be used like multidimensional arrays as well.** Array lists would be more appropriate for matrices that receive regularly updated data. Numpy ndarrays are often used for matrices and vectors in business analytics programs.

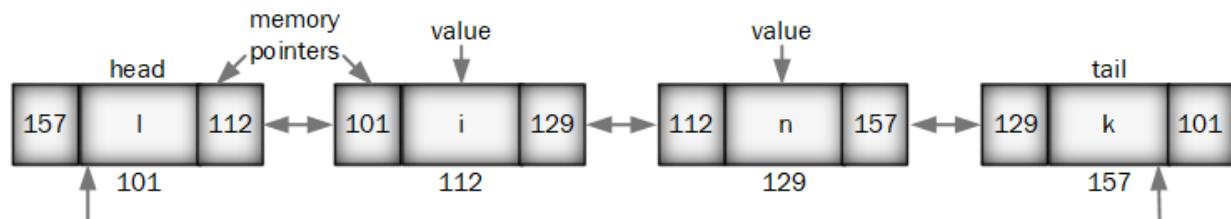
In Python, you can convert a number of different data structures to lists. For example, you can convert the keys or items of a dictionary into a list of keys or items. This can be accomplished by passing the dictionary or dictionary items into the built-in `list()` method. You will learn about dictionaries in an upcoming section. Dictionaries don't have sequential indices like lists. If you ever need a dictionary in sequential order, you must first convert it to a list.

## Linked lists

**LinkedLists** are another implementation of a list. LinkedLists are included by default in languages like Java, but not in Python. A linked list is resizable like an array list. However, linked lists provide faster control over where new list items are added, such as in the middle of the list. Linked lists do not store items contiguously in memory, which means they can be slower than arrays or dynamic arrays for basic operations such as retrieving data. However, they can be faster at inserting and removing items. Each item in a linked list has its own location in memory. Again, these speed differences are often negligible.

Items in a linked list are linked together by storing the memory locations of other items in the list. In a doubly linked list, which is how Java implements linked lists, each item is linked to the item just before and just after itself in a chain. Linked lists start with a head where the first value is stored and end with a tail where the last value is stored. Circular linked lists create a link between the first and last elements of the list. Figure 4.3 shows a representation of a circular (i.e., head points to tail and tail to head) doubly linked list. Many implementations for linked lists exist. Figure 4.3 shows just one such example.

**Figure 4.3: Linked Storage of Items in a Circular Doubly Linked List**



Linked lists can be turned into multidimensional data structures as well by embedding linked lists or other data structures in a linked list, or designing a custom linked list with pointers to items within the matrix structure. Since Python does not possess a built-in linked list, we will not show an example here.

## Dictionaries

Dictionaries, maps, and associative arrays are names for a similar and often the same data structure. For example, Java uses the term map and Python uses the term dictionary to represent essentially the same data structure. Maps and dictionaries map a **key** to a **value**. For example, you could create a key called "name" and set the value of that key to "Jeff." Or, you could have a key called "weight" with a value of 65. **Keys can be used to store data and retrieve data much like array indices.** Unlike arrays, which require sequential integers for indices, maps and dictionaries use keys that can be integers (they can be non-sequential) or strings. Often, keys are strings. Dictionaries are generally as fast at accessing items as arrays and as fast at adding or removing items as linked lists. In other words, dictionaries are generally quite efficient data structures for basic functions such as adding and removing elements.

In a dictionary, each key must be unique. This requirement exists because dictionaries are usually implemented with hash tables. The use of hash tables also explains why they are generally so fast and efficient. A **hash** is a function that quickly converts a key (e.g., name, age, etc.) into a hash value that is: 1) unique (theoretically speaking), 2) always the same given the same key, and 3) a consistent size.

Behind the scenes, a key is converted into an index within the hash table through the use of a hash function. The value of the key is then stored at that particular index in the hash table. Don't worry if that didn't make perfect sense. Really, you just need to know that dictionaries are generally fast and efficient data structures and that they rely on keys to store and retrieve values instead of sequential indices. The following code shows the use of dictionaries in Python.

### Declaring, Instantiating, and Initializing Dictionaries

In Python, a dictionary is created with curly brackets: `groceryList = {}`. A dictionary can be initialized with key:value pairs with a colon separating each key and pair: `groceryList = {"bananas": 6, "carrots": "one bag"}`. Notice in this example, that the values in a dictionary do not need to be the same data type in Python.

Let's assume that you want to code a shopping list in which the dictionary key is the name of the grocery item and the value is the quantity of the item that you want to purchase.

You can add elements to a dictionary as follows: `groceryList["grapes"] = "one bag"`. In this example, "grapes" is the key and "one bag" is the value. You can also remove specific values from a dictionary by calling the key of the value you wish to remove and using the **del** keyword. For example, `del groceryList["bananas"]` would remove the value with the key "bananas" from the dictionary. The code below shows several dictionaries with the add and remove methods.

To get the value stored at a particular key, you simply call the specific key:  
`print(groceryList["bananas"])`.

For a list of more methods you can call on a Python dictionary, see:  
[https://www.w3schools.com/python/python\\_dictionaries.asp](https://www.w3schools.com/python/python_dictionaries.asp).

```
class DictionaryExample:
```

```

def executionDictionaryExample(self):

    #a dictionary of string keys to integer values
    groceryList = {"bananas": 6, "soda": 12, "crackers": 1}

    groceryList["apples"] = 7 #adds the value 7 at key apples

    del groceryList["soda"] #removes the value at key "soda" (i.e., 12)

    #an empty dictionary to store keys and associated values
    jobSalaries = {}

    #initializing key value pairs in the empty dictionary
    jobSalaries["Cashier"] = 22500.00
    jobSalaries["Programmer"] = 70000.00
    jobSalaries["DataScientist"] = 95000.00

    print(jobSalaries["Programmer"]) #would print 70000.00

```

## Data Structures and Classes/Objects

In most of the examples presented to this point, arrays, lists and maps have been used to store primitive data types. However, these data structures can be used to hold more complex data structures than primitive data types. You saw an example of this in relation to matrices. A matrix (i.e., multidimensional array) is simply an array that contains other arrays. You can also create an array to store a dictionary, a dictionary that stores arrays, an array of lists, or a list of arrays.

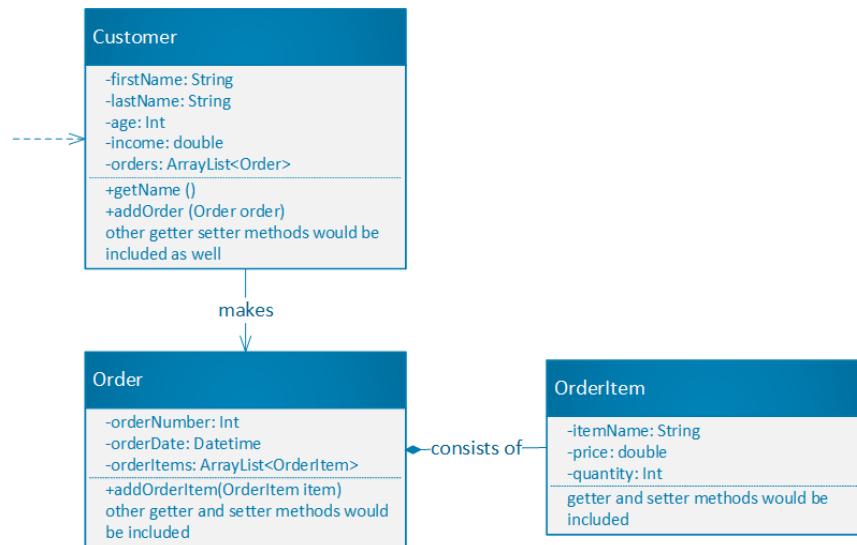
Although these different combinations might be useful in some use cases, it is often advantageous to store objects instantiated from a class within arrays, lists, or dictionaries. Similarly, attributes of an object can store many types of data, such as primitive data types, arrays, lists, dictionaries, or even other objects. Thus data structures can be mashed together to create complex structures.

Let's consider an example. Imagine that you work for a company that is developing a new customer relationship management system (CRM) to help other organizations keep track of their customers. Based on discussions with potential clients, you know that customer service personnel want the ability to see all of the orders made by particular customers. To store information about each customer, we might start by creating a Customer class to instantiate Customer objects that can store information about particular customers. Assume as well that we want these Customer objects to store Order objects that hold information about the orders made by a customer (i.e., purchase history).

Given that a customer can make multiple orders, objects instantiated from the Customer class would need a way to store multiple Order objects. Similarly, an Order is made up of multiple Items (i.e., products). So the Order class would need a way to store items that might be represented by an OrderItem class. Although this program is getting more complex, it is still quite simple in comparison to systems

used in industry. Let's examine what a simple version of this system feature might look like. Figure 4.4 provides a class diagram to show the structure of this simple program.

**Figure 4.4: A Class Diagram Depicting the Example Program**



### Data Structures and Class/Objects

The program that follows is based upon the class diagram in Figure 4.4.

```

class Customer:
    def __init__(self):
        #The orders attribute contains a list that can be filled with Order objects
        self._firstName = None
        self._lastName = None
        self._age = None
        self._income = None
        self._orders = [] #an empty list

    #other attributes about a customer could be added as well

    #create getter/setter functions for each attribute
    def getName(self):
        return self._firstName + " " + self.lastName

    #other getter/setter methods would go here. They are not included for brevity.

    #this method accepts an Order object and adds it to the list stored in the orders attribute
    def addOrder(self, order):
        self._orders.append(order)
  
```

```

#this method retrieves an Order object from the orders list at a specific index
def getOrder(self, index):

    return self._orders[index]

class Order:

    def __init__(self):

        #the orderItems variable will contain a list of OrderItem objects
        self._orderItems = [] #an empty list that will be added to

        #the other class attributes would need to be added here
        #getter and setter methods would go here

    def addOrderItem(self, item):

        self._orderItems.append(item)

    def getOrderItem(self, index):

        return self._orderItems[index]

    #other Order methods, such as calculateOrderTotal() could go here

class OrderItem:

    def __init__(self):

        #the attributes of an order item would go here. Only one attribute is included for
        brevity
        self._itemName = None

    #getter and setter methods would go here
    #other OrderItem methods, such as calculateItemTax(), could go here

#The execution of the program starts below. Everything above is just setting up classes

#create a Customer object for each customer you want to track
customerJeff = Customer()
customerJen = Customer()

#set the attributes for each customer. Only one partial example is provided below.
customerJeff.setFirstName("Jeff")
customerJeff.setLastName("Wall")

#instantiate the orders that Jeff Wall made and set its attributes
#for brevity, not all attributes are set for each Order
jeffOrder1 = Order()
jeffOrder1.setOrderNumber(256)

```

```

jeffOrder2 = Order()
jeffOrder2.setOrderNumber(257)

#create the OrderItems in each order and set the item attributes.
#OrderItems are only shown for one Order and not all attributes are set for brevity
jeffOrder1Item1 = OrderItem()
jeffOrder1Item1.setItemName("Socks")

jeffOrder1Item2 = OrderItem()
jeffOrder1Item2.setItemName("Shoes")

#Add the OrderItem objects to its respective Order using the addOrderItem() method of the Order
object

jeffOrder1.addOrderItem(jeffOrder1Item1)
jeffOrder1.addOrderItem(jeffOrder1Item2)

#Add the Order objects to its respective Customer using the addOrder() method of the Customer
object

customerJeff.addOrder(jeffOrder1)
customerJeff.addOrder(jeffOrder2)

#if you wanted the orders for the Customer "Jen", you would repeat the actions here
#You will learn in later chapters how to automate such efforts through loops and user interfaces

#you could then create a list to store all of the customers
customers = []

#add the customers to the ArrayList
customers.append(customerJeff)
customers.append(customerJen)

#if it is not yet clear to you, in this example, we have created a complex data structure
#we have a list (i.e., customers), that contains Customer objects
#the Customer objects contain a list of Order objects
#the Order objects inside the Customer objects contain a list of OrderItems objects

#if you now wanted information about the first OrderItem (i.e., index 0) that
#Customer Jeff (i.e., index 0) made in Jeff's second order (i.e., index 1), you could use the
#appropriate list indices to get the information

firstItemSecondOrderForJeff = customers[0].getOrder(1).getOrderItem(0)

#You start with the outer list (i.e., customers) and then move inward with the embedded data
structures
#The first index represents a Customer, the second index represents an Order, and the third index
#represents an OrderItem

```

Embedding data structures within data structures, as in the example above, will allow you to create increasingly complex systems. Of course, the code above is still quite simple and not something that you

would use in practice. In practice, the program would include a user interface where a user could input Customer, Order, and OrderItem information through forms or other devices instead of the information being directly coded into the program as in the example above. There would also need to be a connection to a database where customers, orders, and order items would be stored.

It takes time and patience to learn new languages and programming logic and architecture. Remember to be patient with yourself and rely on others for support when needed. Despite the crudeness of the example, it still shows the importance of and power in being able to embed data structures within one another. The example shows how to store lists in object attributes and how to store objects inside of lists. Understanding these foundational principles will help you develop or oversee the development of more complex programs and systems.

## Introduction to Complex Data Structures

To this point, you have been introduced to foundational data structures, namely classes/objects, arrays, lists, and dictionaries. Although these data structures are useful and widely used in practice, there are other data structures that possess beneficial properties that other data structures do not.

First, we will examine a “data frame” data structure that is utilized by a number of analytics and machine learning models in Python. The data frame data structure is part of a Python library called pandas. We will also discuss two advanced data structures herein, namely trees and graphs. Books about data structures go into far greater detail about how to build and use tree and graph data structures. For our purposes, you simply need to understand what the data structures are and how they can be used. Many advanced data analytics models and algorithms rely on tree and graph data structures.

### ***Data frames in pandas***

The pandas library in Python is a popular library that provides tools for managing datasets for analytics and machine learning purposes. The pandas library can be imported into a Python program by calling: “import pandas” or “import pandas as pd”. Once the library is imported, you can use the various methods provided by pandas. Later chapters will explore the functionality of the pandas library in greater depth. For now, we will focus on the underlying data structure in pandas, the data frame.

A **data frame** in the pandas library is a two-dimensional, tabular data structure consisting of rows and columns. It is similar to a matrix (i.e., a multidimensional array). In many ways, you can think of it like an Excel spreadsheet. The two dimensions of a data frame are the dataset’s features (i.e., columns) and records (i.e., rows). A dataset typically consists of a set of features (e.g., height, weight, age, salary, etc.) that describe the subject of the data and data records that store feature values for each given subject.

For example, if you were an accountant examining a dataset of assets managed by your company, the dataset might include features such as: asset name, asset description, expected life of the asset, cost of the asset, how much the asset could be sold for (i.e., salvage value), etc. These features of the dataset would represent the columns of the pandas data frame. The records or rows within this dataset would be values for these features for each asset owned by the company. In the table below, the features run along the top row (i.e., the columns) and the records are represented by the bottom two rows.

Asset Name	Description	Lifespan (years)	Cost (dollars)	Salvage Value
Desk	Wood desk	7	449.99	150
Chair	Leather chair	6	269.99	75

Data frames in pandas include a unique identifier (i.e., index) for each row, so that every record can be referenced when needed. If your dataset does not have a unique id for each row in the dataset, pandas will assign sequential index values as the id for each record (i.e., 0, 1, 2, 3, 4, etc.). Each record in the data frame can be retrieved by its id/index. The table below shows the index values for a data frame.

	Asset Name	Description	Lifespan (years)	Cost (dollars)	Salvage Value
0	Desk	Wood desk	7	449.99	150.00
1	Chair	Leather chair	6	269.99	75.00

Each feature/column in a pandas data frame also has an associated data type. In the example above, the asset name column would have a data type of “object” because the record values for “asset name” are string objects. Description would also be an “object” data type. Lifespan would likely be typed as an integer data type. Cost and salvage value would be typed as floats because they contain decimal values. You should always check the data types of each column in the data frame to make sure that information was imported from your dataset into the data frame correctly. Sometimes numerical columns will be converted to string objects when you import your dataset into a data frame.

The actual code to create a pandas data frame, import data into the data frame, and manipulate data are covered in later chapters. For now, recognize that datasets used for analytics or machine learning purposes can be converted to pandas data frames to allow easy manipulation and visualization of the data.

## Tree data structures

Tree data structures are valuable and essential data structures that are used across many types of programs. Even the program that interprets the code you write relies on tree data structures. The syntax of your code is stored within tree data structures to ensure that the code is executed efficiently and in the correct order. These trees are called syntax trees. Some databases rely on trees to index data in the database to improve the speed of data retrieval. A common tree used for database indices is the B-tree. Trees can also be used to sort ordered data, such as the binary search tree depicted in Figure 4.5. If you plan to pursue a career in data science or data analytics, you ought to learn more about different tree data structures.

Tree data structures are so named because they look a bit like upside down trees as in Figure 4.5. Trees consist of nodes hierarchically interconnected by edges that branch down from the root node. A **node** is an object that stores data values. An **edge** represents a connection between two nodes. If you have two interconnected nodes, the node closer to the root node is the parent node and the other is the child node. To be considered a tree, the relationships between nodes must follow certain rules, such as:

- A tree should have a single root node that acts as the base of the tree.
- Each node should only have one parent node.
  - Nodes should not connect in a cyclic fashion
- A node should not connect to itself.

Figure 4.5 shows what is called a binary search tree, because each node has no more than two child nodes. Although binary trees set limits on the number of child nodes a parent node can connect to, trees do not have to impose such limitations. You can design a tree that has parent nodes with any number of child nodes. Let's examine the binary search tree structure in Figure 4.5 to better understand the utility of the tree data structure. A **binary search tree** has a few unique properties:

- As a binary tree, each node can have no more than two child nodes (i.e., a left child node and a right child node).
- As a search tree, the values stored in the nodes are ordered for fast search and retrieval (e.g., 1, 2, 3, 4, 5).
- The value stored in a left child node must be less than the value stored in the parent node.
- The value stored in a right child node must be greater than the value stored in the parent node.

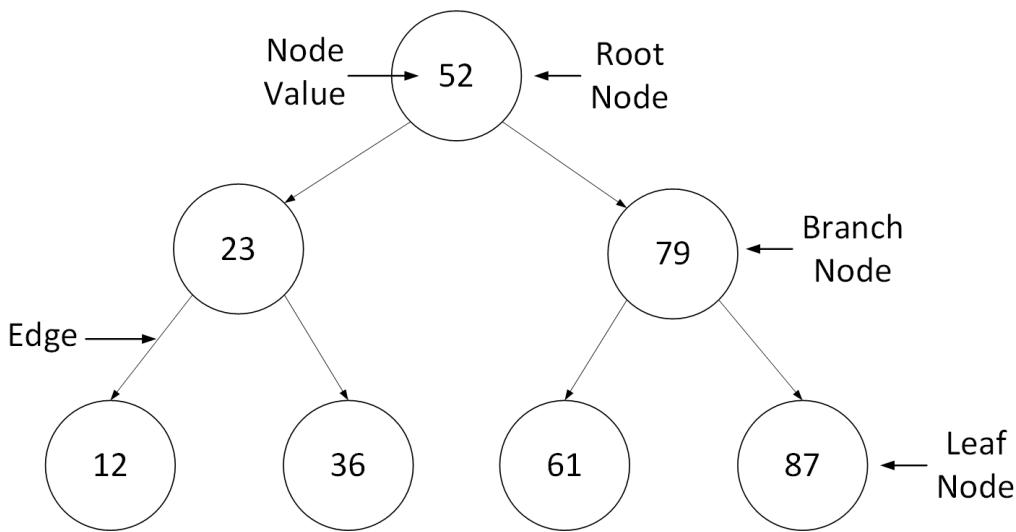
With these properties, you can quickly find an ordered value in a collection of values. Each branching of the tree essentially halves the distance between the next number.

Consider for example that you want to find the node with the value 61. You would start at the root node, which stores the value 52. Based on the properties of the binary search tree, you know you can ignore the entire left side of the tree because 61 (i.e., the number you are searching for) is greater than 52. As such, you would move to the right node with 79 as the value. Because 61 is less than 79, you know you can ignore the right child node of the node with the value 79. Next, you would move to the left node with 61 and find your answer.

By making three moves in the data structure, you were able to arrive at the correct node and retrieve the value. With an array that contained pre-sorted data, you would have to iteratively check the following values: 12, 23, 36, 52, 61 before arriving at the appropriate value. Although that is only 5 movements, the number of movements can grow substantially as the size of your data set increases. Further, the array example assumes the array was pre-sorted, which may not be the case in the real world. Pre-sorting data requires time for computation. In an unsorted array, the value 61 could have been at the beginning, end, or anywhere in between, creating inconsistency in the number of movements required to find the number. From this simple example, you can start to see the value of trees.

Binary search trees are also designed to insert new values into the structure efficiently. To insert a value into an array, you would have to iterate through much of the array, find the appropriate location, add the new data value and then move every value thereafter. The binary search tree has many useful properties.

**Figure 4.5: Depiction of a Binary Search Tree Data Structure**



Other types of trees possess different properties that are designed for specific applications. Although trees can be very different in design, they often share a common set of operations (i.e., methods). Trees will possess an **insert()** method that allows you to add a node, a **remove()** method that allows you to remove a node, **traverse()**, and often a **search()** method.

**Traversing** is the act of visiting each node in a tree. Traversing a tree is similar to iterating through the indices in an array. There are a number of ways to traverse a tree. Broadly, two of the most common methods are depth-first search and breadth-first search. **Depth-first search** moves through the nodes by exploring the depths of branches before moving to other branches. For example, one form of depth-first search would move from 52 to 23, then from 12 to 36 in Figure 4.5. It would then jump to the other side of the tree to 79, then 61 and 87. **Breadth-first search** traverses the tree by level. For example, a breadth first search would start with 52, move to the second level visiting 23 and 79, and then move to the last level visiting 12, 36, 61, and 87. Each type of traversal method serves a different purpose, such as copying a tree and reading the contents of a tree. If the tree is designed for searching for specific values, it will also possess a **search()** method to find a specific value.

There are many tutorials that will walk you through how to create a tree from scratch. There are also many libraries written by other developers that you can use to implement trees in your programs. For example, a number of well-known analytics libraries in Python provide support for different types of trees, such as the SciPy and scikit-learn libraries. From an analytics perspective, trees are used for a variety of reasons. We will examine one such tree in the next section, decision trees.

### **Decision trees for predicting business outcomes**

**Decision trees** are a machine learning algorithm that rely on the tree data structure to encode decision rules into a hierarchical structure to predict an outcome. **Machine learning** is a branch of computer science that often seeks to mimic human learning. Machine learning models, such as decision trees, attempt to learn patterns in datasets to predict future outcomes. For example, a decision tree could be used to determine whether a stock price will go up or down based on patterns that the tree algorithm identified from a data set with information about the company, market conditions, etc. It could also be

used to determine whether to hire a certain employee based on attributes of the employee given a data set of previously hired employees.

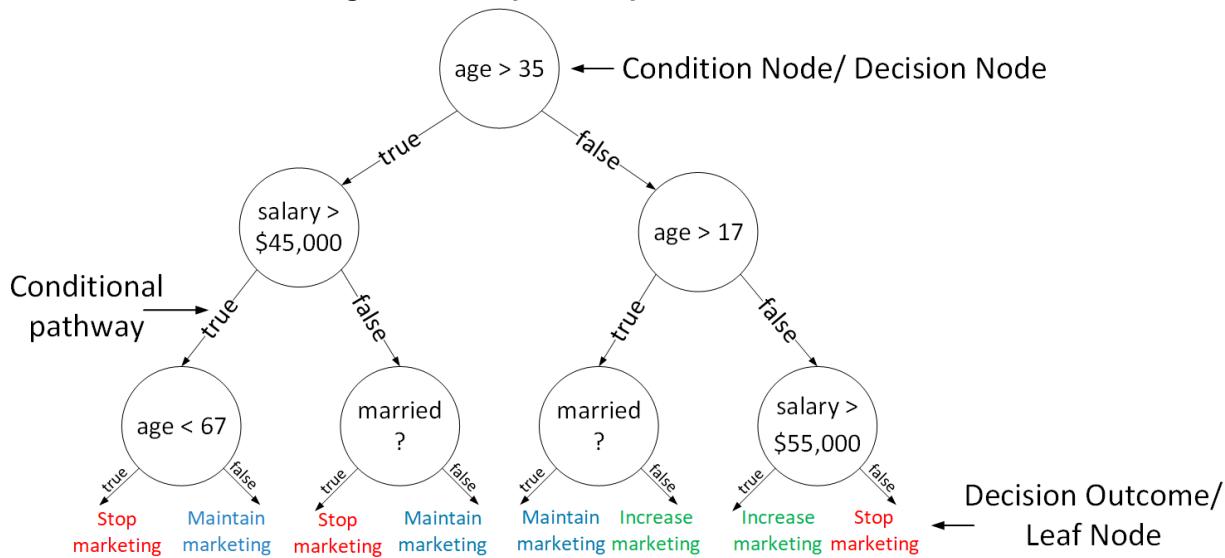
Decision trees consist of nodes and edges similar to other tree data structures. Decision trees tend to be binary trees like binary search trees (i.e., only two child nodes per node). Figure 4.6 depicts a simple decision tree that a marketing team might use to decide if they should: 1) stop sending marketing messages to an individual, 2) maintain current levels of investment, or 3) increase marketing investments (i.e., the possible outcomes). The decision tree model would learn patterns about ideal decisions by extracting patterns from a dataset of previous marketing investment decisions.

In this particular example, the outcomes are called classes (not to be confused with classes in object-oriented programming). The decision tree tries to find the appropriate class (i.e., stop, maintain, or increase marketing investment) for new decisions that must be made based on a set of features in the dataset. This type of decision tree is known as a **classification decision tree**, because it selects outcomes from a set of predefined classes (e.g., stop, maintain, or increase marketing investment). Decision trees can also predict continuous values, such as predictions about how much profit could be gained or loss from a marketing investment decision. Such decision trees are known as **regression decision trees**.

Decision trees predict the appropriate outcome class or continuous value based on a set of features. The features in Figure 4.6 include the age of an individual, the salary of the individual, and whether the individual is married. **Features** are simply attributes of a dataset that can vary. Decision trees store feature-based conditions within each node of the tree. Decision trees learn the best configuration of feature-based conditions from data sets. When you want to make a prediction/classification for a new data point, the learned set of conditions can be used to predict/classify the outcome of the new data point.

Figure 4.6 assumes that the decision tree has already been trained on a dataset of past marketing decisions. The feature-based conditions in each node represent what the tree learned from the data set. For example, the root node in Figure 4.6 presents a feature-based condition that asks whether an individual's age is greater than 35. This feature-based condition would have been set as the decision tree algorithm explored the dataset of prior decisions to find the best way to classify outcomes. The left and right edges of each node often represent true/false or yes/no answers to the node condition. Based on the rules learned in the tree in Figure 4.6, the decision tree would predict that the marketing team should maintain marketing investment levels if an individual is older than 35 (i.e., move left from the root node), earns more than \$45,000 (i.e., move left from the salary > \$45,000 node), and is more than 67 years old. You could use the learned model to predict investment decisions for multiple future individuals.

**Figure 4.6: Simple Example of a Decision Tree**



So, how do you determine what conditions should exist at each node and how the set of conditions lead to specific outcomes? Again, decision trees can be trained from data. Decision trees are an example of **supervised learning models**, as they require existing labeled data (i.e., data with known outcomes) to supervise the training (i.e., learning) of a predictive model. The training process helps to identify the appropriate conditions, the placement of those conditions within branches of the tree, and the outcomes associated with a particular traversal through the nodes. Again, this training process is determined by a dataset as the model learns patterns from the dataset. There are many different algorithms that can be used to build the decision tree from a dataset. Texts about data mining and machine learning go into greater detail about these algorithms and how they work. For now, simply recognize that tree data structures can be used in machine learning to help managers and organizational employees make decisions.

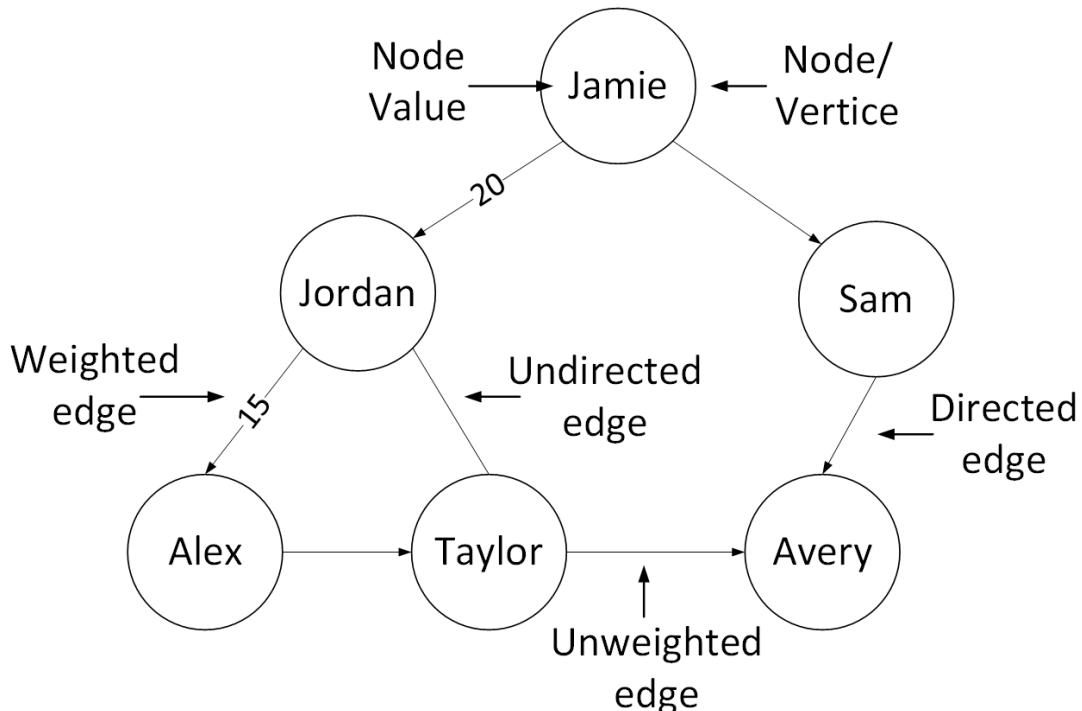
Although decision trees are a popular machine learning model, training decision trees can lead to widely different models given slight variations in data values in the dataset. Decision trees are quite sensitive to variations in data. To combat this, multiple decision trees may be combined together into an algorithm known as a random forest (a.k.a., random decision forest). **Random forests** take the outcomes of many different decision trees to make predictions. For example, a subset of features for each decision tree in the “forest” might be selected randomly from the set of all features. The prediction outcomes of all of the individual tree models are then combined in some fashion, such as by averaging, considering the mode outcome, or through more complex methods. Random forests are an example of an ensemble model. **Ensemble models** merge together multiple models by finding the mean, weighted mean, or some other aggregation metric of the models that make up the ensemble. Many Python libraries exist, such as the sklearn library to build and train decision trees and random forests using datasets. These libraries can accomplish this work with only a few lines of codes.

## Graph data structures

Graphs are another data structure that has increased in popularity. In some ways, graphs are similar to trees. Like trees, graphs consist of nodes (sometimes called vertices) and edges. Vertices (a.k.a., nodes) in graphs are used to store data similar to the way trees store data. Edges represent connections

between nodes. Edges can be weighted with some value to represent the strength or importance of the connection between nodes. Despite the similarities between graphs and trees, graphs tend to be more flexible and have less rigid rules than trees. If you recall, trees are hierarchical with strict parent-child relationships that don't support cyclical or bidirectional connections between nodes. The concepts of hierarchy and parent-child relationships don't necessarily apply to graphs. Figure 4.7 shows a graph depicting relationships between people. To fit a variety of concepts into one graph, the graph shows some edges as directional and others as undirected (a.k.a., bidirectional). This type of graph is known as a mixed graph. Generally, a graph will either be weighed or unweighted, so the example is not necessarily true to form.

**Figure 4.7: Simple Example of a Decision Tree**



Graphs have grown in popularity in computing with the advent of social media. In a social media application, one user can be connected to many other users in a tangled web of connections. Tree data structures don't make as much sense for a situation like that. In social media, graphs can provide valuable insight into who influential members of a community are. They can also identify important individuals who act as bridges between different communities. This information can be useful as managers try to identify human resources for projects. Social graphs can also be used in marketing to find influencers who will be the most effective in reaching a target market. Simple graphs can be stored in maps/dictionaries. It is also possible to create more complex graphs that contain more information within a node using classes/objects to represent nodes.

#### A Simple Graph Implementation

The following dictionary stores the information in Figure 4.6, except the edge weights. Classes/objects would be required to store weights, or possibly a dictionary of an array of

dictionaries. Simply put, to store more information like weights, a more complex data structure would be needed. To keep the example simple, we will ignore the weights.

The dictionary consists of lists of connected nodes as the value for each key. The keys of the dictionary represent each of the nodes. In this case, each node represents a person in a social network. The values in the list stored at each key represent the node connections that the node links to (i.e., the edges).

```
graph = { "Jamie" : ["Jordan", "Sam"],  
          "Jordan" : ["Alex", "Taylor"],  
          "Sam" : ["Avery"],  
          "Alex" : ["Taylor"],  
          "Taylor" : ["Jordan", "Avery"],  
          "Avery" : []  
      }
```

Graphs require methods to **add()** nodes and edges, **remove()** nodes and edges, **traverse()** methods to visit each node in the graph, and **search()** methods to find specific nodes. If you want to identify the most important nodes in the graph (a.k.a., influencers), you might also want to use methods to calculate the centrality of each node. **Centrality** is a measure of how important or influential a node is within the graph. Centrality can be measured in a number of ways and take on slightly different definitions of “importance” based on the chosen method. Other more complex algorithms exist to measure other properties of graphs. Graph theory is complex and entire courses are dedicated to the topic.

Armed with a foundational knowledge of data structures, you will be able to provide analytics support to your organization. If you wish to develop analytical skills, you should spend some time further refining your knowledge of data structures.

## Chapter 4 Assignment

Please see the [Chapter Assignments Workbook](#) for the Chapter assignment.

# Chapter 5:

## Data Structures and Flow Control

In Chapter 4, you learned about different types of data structures that allow you to store collections of information. You learned how to create different data structures and add and remove data from these structures. You also learned how to access the value of particular items in the data structure by calling an index or key. However, that information isn't particularly useful by itself. More commonly, you will need to iterate through the elements in an array, such as iterating through all transactions made within an account to total the expenditures of the account. To do this, you need to understand how to control the flow of information and logic within your program, a.k.a., control flow. This chapter will show you how to write programs with more complex flow logic.

### Flow Control

Flow control represents the path a system will take to execute the code you write. Recall from the Chapter 3 discussion of breakpoints that statements and method calls are a form of flow control that move the execution of the code sequentially from line to line and method to method. In Python, the simplest form of flow control would be a program without any methods. Classes and methods are not required in Python. You could simply write simple statements with variables that would manipulate data line by line. This code would start with the first line of code, move to the second line, and continue until it executed the last line. Few programs are this simple though.

Information systems are complex and the business rules they encode are rarely static or linear. Business operations consist of choosing from options, making decisions, and repeating certain operations over and over again. Many of these various activities can be encoded into computer code. We live in a time of transition in which organizations are seeking to automate more and more work, including information work. Automation is often accomplished through the design of intelligent information systems. Modern systems exhibit complex flow control logic, which increasingly includes analytics, machine learning, and artificial intelligence models. This text primarily focuses on basic control logic, such as if() statements, that allows you to perform conditional logic and various forms of loops to repeat actions.

### ***Programming optional control flows***

Sometimes you need to include the ability in a program to choose whether or not to perform a certain action (i.e., execute an option). For example, if you were writing a program to allow customers to design a custom vehicle, you could provide several design options to the customer. Some of the options would really be decisions between alternatives (e.g., car vs. truck vs. SUV), which are examined in the next section. Others would be true options, such as adding a trailer hitch or not or including heated seats or not. Lets begin with these boolean types of options (i.e., include or don't include, do or don't do, true or false).

An option can be encoded with an if() statement. An **if() statement** checks if a condition is true. If the condition is true, the code within the body/block of the if statement is executed. If the condition is false the code in the if statement block is ignored and the flow of control moves to the next statement in the

program. A condition in an if statement is checked with a comparison operator. Comparison operators simply compare two values in a specified way.

**Common comparison operators include:**

- equal to: if ValueA == ValueB is true perform the if-block logic
- not equal to: if ValueA != ValueB is true perform the if-block logic
- greater than: if ValueA > ValueB is true perform the if-block logic
- Less than: if ValueA < ValueB is true perform the if-block logic
- greater than or equal to: if ValueA >= ValueB is true perform the if-block logic
- less than or equal to: if ValueA <= ValueB is true perform the if-block logic
- Modulus operator (does a remainder exist if the values are divided): ValueA % ValueB

You can combine or chain multiple conditions together using "and" and "or" operators within an if() statement. For example, the code inside an if() block with an "and" operator and two conditions would only run if both conditions were true:

```
if valueA == 6 and valueB == 4:  
    #code to execute if both conditions are true.
```

Conversely, the code inside the if() block with an "or" operator and two conditions would run if at least one of the conditions were true:

```
if valueA == 6 or valueB == 4:  
    #code to execute if at least one condition is true.
```

The following code shows how to manage the flow control for options in Python.

### If Statements for Options

The code below shows some uses for the if() statement in Python. The first if() statement checks to see if the variable addHeatedSeats is set to true. If it is true, the addHeatedSeats() method is called; if not, nothing happens. The second if() statement checks to see if the vehicle price is greater than \$15,000 or if the customer is from Houghton, MI. If at least one condition is met, the customer receives a free upgrade on the vehicle.

In Python, the "and" operator is represented by the word "and", and the "or" operator is represented by the word "or".

```
class VehiclePurchaseHandler:  
  
    def handlePurchase(self):  
  
        #in a real program, the values below would be passed in and not hard coded  
  
        customer = Customer()  
        customer.city = "New York" #assumes the city attribute is public
```

```

customerVehicle = Vehicle()
customerVehicle.price = 20000.00 #assumes the price attribute is public

addHeatedSeats = True

#the if() statements

if addHeatedSeats:

    #/this would execute because addHeatedSeats is set to True
    customerVehicle.addHeatedSeats()

if customerVehicle.price > 15000 or customer.city == "Houghton":

    #this would execute because at least one of the conditions was met
    customerVehicle.addFreeSatelliteRadioSubscription()

```

If you choose to learn more about programming, explore texts and courses that teach you about design patterns. Many if() statements can be avoided through the use of advanced design patterns, such as the decorator pattern. However, if statements will help you build simple analytics programs and are found commonly in business programs, particularly business analytics and machine learning programs.

## **Programming alternative control flows**

Sometimes, you will need to represent choices between different alternatives within a program instead of just whether to include an option or not. If() statements are also used to encode **choices between alternatives** into a program. Each alternative is represented by an if() statement, an else if() statement, or an else statement. The if() statement is the first alternative to be checked against. After the first if() statement, other alternatives will be represented by elif() statements (i.e., else if) or by an else statement.

**If your business case has a default alternative**, it should be represented by the else statement. The **default case** is the alternative that is selected if all other conditions for the other alternatives evaluate to false. Excepting the first if() statement and the default "else" case, all other alternatives will be represented by elif() statements. Elif() stands for else if. So you start with an if statement, continue with else if statements, and end with an else statement.

If you know the probability of each alternative being selected, you should place the alternative with the highest probability of being selected in the first if() statement. Doing so can minimize the number of conditions that the program must check. Modern computers are fast and often don't require this level of scrutiny though. Yet, some business applications need to perform quickly to respond to real time shifts in markets. In these cases, developers look for any way they can find to trim milli-seconds from program execution time. For example, automated high-frequency stock trading relies on gaining advantages through time advantages by utilizing advanced hardware and software.

To explore the concept of if() statements to represent alternatives, assume you are writing a program to incentivize employees through the gamification of work. **Gamification** is the use of game and video game concepts, such as points, rewards, missions, etc. to make work more playful and enjoyable to employees. Assume the program helps managers keep track of points earned by employees and rewards employees

can receive based on how many points they score. As employees gain more points through work tasks, larger rewards can be earned. Suppose that employees with 10-14 points can earn a branded sweatshirt, employees with 15-19 points can earn a branded sweatshirt and baseball cap, and employees with 20+ points can earn a branded sweatshirt, baseball cap, and pair of sunglasses. Also suppose that everyone who receives less than 10 points receives a branded mug just for playing (i.e., the **default alternative**). The code below shows how this could be accomplished with if() statements.

### If Statements for Alternatives

The code below shows how the example above might be implemented in Python. In Python, you begin a set of alternatives with an if: block followed by elif: blocks and an else: block if a default value is needed. Notice that else-if blocks are labeled elif in Python.

```
class RewardTracker:

    def track(self, points):

        if points >= 10 and points <= 14:

            prize = "branded sweatshirt"

        elif points >= 15 and points <= 19:

            prize = "branded sweatshirt and baseball cap"

        elif points >= 20:

            prize = "branded sweatshirt, baseball cap, and pair of sunglasses"

        else:

            prize = "branded mug"

        return prize

#Instantiate and run the RewardTracker class

tracker = RewardTracker()
employee1Reward = tracker.track(15) #would return "branded sweatshirt and baseball cap"
employee2Reward = tracker.track(9) #would return "branded mug"
```

In some languages like Java, alternatives can also be represented by switch-case statements. Python does not have a built-in equivalent to switch-case statements. **Switch-case statements** are used to set different behaviors for different cases. Thus, switch-case statements test for equality. Switch-case statements are not as useful for situations where your conditions include greater than, less than, modulus, or other operators for condition checking. However, some developers prefer the format of switch-case statements in place of if() statements when writing flow logic for alternatives. The switch() portion of the switch-case statement is passed a variable holding an integer value that represents a specific case (i.e., 1 for Sunday, 2 for Monday, etc.). Inside the switch block/body, different cases are checked. If the value

passed to the switch() statement matches the value of a particular case, the code within that case block is executed. Switch-case statements can also have a default alternative if all other cases evaluate to false.

For example, assume you are a highly biased manager who bases employee bonuses on how much you like your employees on a personal level. You could write a program to encode your biased bonus payouts and then tell your employees that a complex computer algorithm was used to generate the values. If it isn't clear, you should NOT do this! For an interesting read about how poorly reasoned management decisions are made through computer programs and data analytics, read [Weapons of Math Destruction](#) by Cathy O'neil. The code below shows how you would write switch-case statements in Java. Don't worry too much about the example since we are learning Python and it does not include switch-case statements.

### Switch-Case Statements for Alternatives in Java

The code below shows how the example above might be implemented in Java. Notice the break statement after each case. The break statement tells the program to stop executing if a particular case evaluates to true. Without the break statement, the default case would be executed each time as well. In the particular example below, that means that bonus would always be 10.00. In this example, we don't want that behavior. With that said, the break statement is optional. Sometimes you may want to perform an action for a case, but also perform the default option regardless of whether other cases evaluate to true or false.

```
public class Main {  
    public void main() {  
        int employeed = 1; //values for each employee would be passed in  
        double bonus; //declare the bonus variable  
        switch(employeed) {  
            case 1: bonus = 10000.00; //you like employee 1 the most  
            break; //stops execution of the switch if the employeed is 1  
            case 2: bonus = 1000.00; //you like employee 2 more than most  
            break; //stops execution of the switch if the employeed is 2  
            case 3: bonus = 5000.00; //you like employee 2 a lot  
            break; //stops execution of the switch if the employeed is 3  
            default: bonus = 10.00; // you don't like the rest of your employees  
        }  
    }  
}
```

}

If() statements are an integral part of most business programs. For example, assume you are writing a system for your employees to access highly proprietary product development information. Assume that you don't want every employee to have the same access to the data, so you create different access rights depending on an employee's job role. When an employee tries to access the data you must check if() the employee's access rights allow them to view the information. Alternatively, consider that you are writing an investment program to help you make automated investments. Suppose you want to change your trading strategy based on if() interest rates are above or below some threshold value. In these and others cases, if() logic or similar logic is required to help you accomplish your objectives. At its core, if() logic is business logic. Business processes include many if(), else-if(), and else conditions (i.e., decisions) that an automated system must take into account.

As a business or systems analyst, part of your job will be understanding business decision points to help yourself or a team of developers automate the decisions. Be prepared to think in terms of and encode options and decision alternatives within programs.

## ***Looping for repetitive control flow***

Options and alternatives implemented with if() statements create conditional logic to guide flow control. Another important aspect of flow control is looping. **Loops** allow you to perform the same action repeatedly until a specified condition is met. Loops accommodate this repetition without you needing to write the code for the action multiple times. For example, assume you needed to calculate the total of an order consisting of 100 different items. If you had to calculate the total manually, you would need to write a lengthy program, such as: itemTotal = item1.price + item2.price + item3.price + ... item100.price. This is problematic for a number of reasons. First, it is tedious. Second, it assumes that you know how many products exist within every order. What if the order consisted of 101 items or 99 items? Your program would be worthless in these cases. By using loops, however, you could add the price of each item in the order until there were no more items to process. Thus, you don't have to write tedious and repetitive code, nor do you have to know beforehand how many items exist in a collection of items (e.g., an array, list, or dictionary).

Programming languages often contain a variety of different types of loops, such as: while() loops, for() loops, and foreach() loops. Each type of loop has a slightly different syntax, but serves the same basic purpose, iterating to repeat tasks.

Loops such as the **for() loop** and **while() loop** use counters so that the loop knows what iteration it is on and when to stop iterating. These loops are most appropriate for data structures like arrays and lists that rely on sequential indices to store information. These loop counters are often used to refer to the indices of an array or list. Often, developers use a variable named "i" (i.e., iteration) to store the current counter value of the loop. Each time the loop finishes an iteration, it increases the counter by 1 and moves to the next iteration of the loop until some condition is met. For example, the for() loop in many languages looks similar to the following code:

```
for(int i = 0; i < 10; i++) { //code to perform on each iteration goes here; will stop when i reaches 10}
```

The loop above would run through 10 iterations. In the example, the first part of the for loop declares a variable called "i" with an integer data type and sets the counter to zero for looping. Again, loops are often used on data structures like arrays, which tend to start with a **zero index**. So, it is common to set "i" to zero (i.e., the first index of an array or list). The next part of the loop sets the condition for stopping the loop (i.e.,  $i < 10$ ). In the example above, the loop should stop before it reaches 10 (i.e., looping from 0 to 9; 10 iterations). Again, you don't want it to reach 10 because the counting starts at 0 instead of at 1. If you have an array with 10 items (i.e., items with indices 0-9) and you try to access index 10, an Exception will be thrown.

The last part of the loop (i.e.,  $i++$ ) increases the value of the counter by 1 after each iteration of the loop is completed. The statement  $i++$  simply tells the program to increment the current value stored in variable "i" by one. If you were to run this loop, the variable "i" would be 0 on the first iteration, 1 on the second iteration, 2 and the third iteration, and so on until it reached 10 and stopped. By doing so, you could access the first 10 indices of an array or list. Recall that list items are accessed by calling the variable name the list is stored in and the index of the data you wish to access: `list[0]` or `list[3]`.

**Be careful when using counter-based loops.** It is easy to get into an infinite loop if you forget to use a counter or use the counter improperly. If you are running your program in the Jupyter IDEs and you have an infinite loop, it may appear as though nothing is happening. If you wait long enough, the program will generally throw an Exception. However, you can also stop the program execution by clicking the stop symbol in the notebook. If you run into this issue, first make sure that your condition is not set to a large number or that you included a condition that can never be met. Second, check to make sure your counter variable is set to increase after each iteration. A counter that does not increase will never reach the stop condition of the loop.

### **Using for and while loops with data structures**

Loops can be used to perform calculations on data structures such as arrays and lists. Performing loops on arrays allows you to execute mathematical functions on vectors and matrices and manipulate collections of objects in a consistent way. Arrays have a size (i.e., the number of items they contain). As suggested above, loops continue until they reach a condition. In the case of looping over an array, the condition is related to the size of the array. You should loop through the array indices until your loop counter reaches the size of the array (i.e., one loop iteration for each array item). In Python, you only want to continue the loop while the counter is less than the size of the array, because arrays in Python start counting at 0 instead of at 1. You will see this in the example below. In Python, the size of an array is accessed by calling the built-in `len()` function and passing the array as an argument (i.e., `len(myArray)` ). The size of a numpy ndarray can be accessed through the `size` attribute of the array (i.e., `myArray.size`).

Let's start with an accounting example, namely depreciation. **Depreciation** is used in accounting to spread the cost of an asset, such as buildings and equipment, over its lifespan. So, instead of buying a building for \$250,000 and showing a \$250,000 decrease in net income in the year it was purchased (that could look like a big hit to profits), companies can decrease net income by a depreciated amount each year that the asset is expected to be used by the company (e.g., \$25,000 per year over ten years instead of \$250,000 in one year).

Assume you regularly purchase computer servers and other IT equipment for your company. Assume you are required to calculate the depreciation value of the equipment before you can have the purchase approved by management. To keep things simple, let's assume your company uses straight line

depreciation (i.e., a simple form of depreciation). The **straight line depreciation** formula is: (asset price - salvage value)/expected lifespan. The salvage value is the amount of money you could get by selling the asset at the end of its expected lifespan. For example, assume you purchase a \$5,000 server with a salvage value of \$300 after a five year period. The depreciated value would be:  $(\$5,000 - \$300)/5$  years or \$940. That means \$940 could be expensed each year over the five-year lifespan of the server instead of expensing \$5,000 at the time of purchase. Calculating one asset is relatively easy. You just plug the numbers into the formula. But assume that you are ordering many IT products for a big infrastructure upgrade. In this case, lists and loops could make your job much easier by allowing you to loop over the items in the list to perform the depreciation calculation for each purchased item. This is essentially how many ERP systems handle such tasks.

### For and while loops with Lists

In the example below, multiple lists are used to represent the parts of the **straight line depreciation formula: (asset price - salvage value)/expected lifespan**. One list represents the asset name, another asset price, another the salvage value, and another the lifespan. Another list is also created to store the depreciated values of each asset. To keep the example simple, assume you need to determine the depreciated value of the following IT assets:

- A server that will cost \$6,500 with a salvage value of \$500 and an expected lifespan of 5 years
- A business-grade router that will cost \$3,000 with a salvage value of \$300 and an expected lifespan of 7 years
- A desktop that will cost \$1,500 with a salvage value of \$100 and an expected lifespan of 3 years

Notice that each list follows the same index order for the respective products. For example, the data for the server is at index 0 in each of the lists, the data for the router is at index 1 in each list, etc. This structure is necessary for calculations to work correctly because we will be looping based on the indices.

The depreciated array would be filled with the following depreciated values after running the loop: 1200.00, 385.71, and 466.66. The while() loop calculates the depreciated values, and the for() loop prints out the values. The print statement in the for() loop could have been added to the while() loop. The for() loop only exists to show you how for() loops operate in Python.

```
class DepreciationCalculator:  
  
    def depreciate(self, asset, assetPrice, salvageValue, expectedLifespan):  
  
        #a while loop uses an external counter variable  
  
        depreciatedValue = [] #initialize an empty array for the results  
  
        i = 0; #the counter for the while loop  
  
        #the condition in the while loop is checked before starting each iteration of the loop  
        #the condition is set to stop when the counter reaches the length of the list  
        while( i < len(asset) ):
```

```

#because indices in each list represent the same asset, we can use i to loop
#i would be 0, then 1, then 2 as it continued through the loop

#each list is simply put into the proper place in the depreciation formula
value = (assetPrice[i] - salvageValue[i]) / expectedLifespan[i]

depreciatedValue.append(value) #add the depreciated value to the list

i += 1 #increase the counter by 1 after each iteration

#a for loop can also be used for looping.
#Notice that we use j instead of i, because i is used previously in the program
#This print statement could have been added to the while() loop, but is used here
#to show you how for() loops work in Python
#Notice that in Python, the stop point is set by calling "in range()"

for j in range(len(asset)):

    print("The expected value of the {} would be {}".format(asset[j],
    depreciatedValue[j]))

#instantiate and run the DepreciationCalculator() class
#notice that the values in the lists are in the same order by product

asset = ["server", "router", "desktop"] #the assets
assetPrice = [6500.00, 3000.00, 1500.00] #the asset prices
salvageValue = [500.00, 300.00, 100.00] #the salvage values
expectedLifespan = [5, 7, 3] #the expected lifespan

calculator = DepreciationCalculator() #instantiate a DepreciationCalculator object
calculator.depreciate(asset, assetPrice, salvageValue, expectedLifespan) #get depreciation

```

**This example could have been simplified to a single array or list** if an Asset class were created with the following attributes: name, price, salvageValue, expectedLifespan, and depreciatedValue. The single array or list could then store three Asset objects with their respective name, price, salvage, and lifespan attribute values. Each of the objects in the array would have its attributes set to the values of one of the three products in the story. Then the depreciatedValue attribute could have been set by calculating the depreciated value for each Asset object. This better object-oriented approach wasn't used in the example so you could learn how to work with multiple vectors and matrices. For many analytics calculations, multiple vectors or matrices are used instead of objects. For practice using arrays of objects, try reducing the example above to a single array or list.

## Using foreach loops

In the previous section, you learned how to iterate over arrays using the while() and for() loops, which use explicit counters. The counters represent the sequential index values of the arrays. In this section, you will learn to iterate over dictionaries using the foreach/for-in loop. Recall that dictionaries can have keys that are strings (e.g., myDictionary["key"] = "hello"). Because the keys are not numeric, you can't easily iterate over them with for() and while() loops because counters are sequential numbers and keys are not. Foreach/for-in loops can also be used for other data structures, such as lists. In some programming

languages these loops are called foreach loops and in other languages they are called for-in loops. Python uses the “for-in” notation.

Let's assume you are tasked with writing a program to calculate the merit raises for employees at your company. Assume that all employees receive a 2% raise on their base salary each year to account for inflation. If you had 1,000 employees, it would be painful and wasteful to calculate each employee's raise manually. Instead, you could write a program to iteratively calculate the salaries. Let's examine how this might be done with a dictionary and a for-in() loop. Please note that you could also use arrays and while() or for() loops or a single loop with Employee objects to accomplish this same task. There are often many ways to program that same result.

### For-in Loops with Dictionaries

In the example below, we first need to store the employee salary data in a data structure. In this case, a dictionary could make a reasonable data structure because you can use an employee identifier for the key and salary data for the value. In the example below, the employee name is used for the key. This would be a bad idea in practice because people can have the same name and dictionaries must have unique keys. Instead, a unique employeeId would likely be used for the key. In practice, you would also likely pull the data from a database instead of writing it manually into the program.

After implementing the data within a data structure, you can then loop over it and multiply the current salary for each employee by 1.02 (i.e., a 2% raise) to get the new salary.

```
class PayRaiseCalculator:  
  
    def calculate(self, salaries, raiseRate):  
  
        #the for-in loop converts the items in salaries into key and value variables  
        #the key and value are updated with new values on each iteration of the loop  
  
        for key, value in salaries.items():  
  
            newSalary = value * (1 + raiseRate) # calculate the new salary  
  
            print("The salary for {} changed from {} to {}".format(key, value,  
            newSalary))  
  
            salaries[key] = newSalary #update the old salary with the new salary  
  
#Instantiate and run the PayRaiseCalculator() class  
#In a real program, this dictionary would be created from database entries  
  
salaries = {"Pat": 50000.00, "Jayden": 60000.00, "Erin": 55000.00}  
  
calculator = PayRaiseCalculator()  
calculator.calculate(salaries, 0.02)
```

If you only wish to loop over the keys of the dictionary, you can alter the for-in loop like this:

```
for key in salaries.keys():
```

If you only wish to loop over the values within the dictionary, you can alter the for-in loop like this:

```
for key in salaries.values():
```

Again, this whole example could have been designed in an object-oriented fashion by looping over Employee objects that contained id, name, salary, and other relevant class attributes.

You have now seen a variety of ways to loop over data structures. With conditional logic and loops, you can create complex flows of control that allow you to account for different options, alternatives, and repetitive work. In later chapters, you will use this knowledge to write simple analytics programs.

### ***Looping with objects***

Most of the looping examples in the previous sections showed examples of loops on data structures containing primitive data types. Understanding how to loop over primitive data types is important for those interested in data analytics. Matrix and vector mathematics, which underlies many analytics and machine learning calculations, relies on primitive data types like integers and floats. However, programming for business systems cannot always be accomplished in such a simple manner. Often, it is more efficient and less error prone to develop business systems with objects that contain multiple attributes instead of breaking these multiple attributes into several different arrays, lists, or dictionaries. Recall that an object is just a custom-designed data structure consisting of the attributes included in the class from which the object is instantiated. The methods of a class primarily exist to manipulate the attributes of the class.

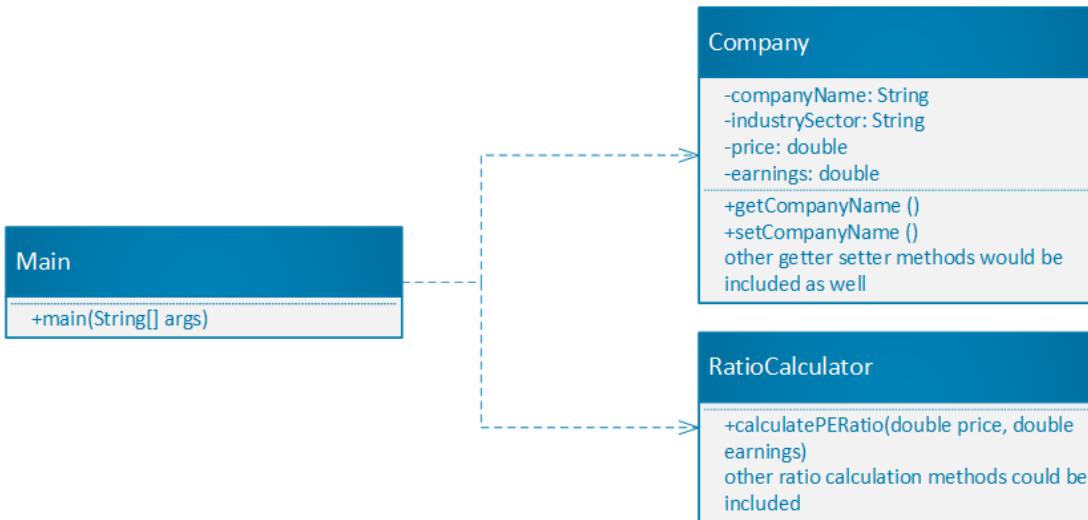
To see how objects and loops can be used together, let's examine a simple example of stock investment. Assume you want to make stock trades based on fundamental analysis. Fundamental analysis seeks to identify the value of a stock based on fundamental information about the company, such as revenue, earnings, stock price, etc. Often, multiple ratios are calculated from fundamental data to determine the value of a stock, such as the stock price to earnings per share ratio (a.k.a., price/earnings ratio or P/E ratio).

One way to approach this problem would be to create multiple arrays to represent different fundamental data features for a variety of company stocks: one array for revenue values, one array for earnings, one array for stock price, etc. The indices of these arrays would represent specific companies. You would need to ensure that index 0 of each array represented the same company, that index 1 represented the same company across the arrays, and so on for each company in the arrays. This approach is error prone though. A missing data value for one of the companies in one of the arrays would have the potential to throw off the indices. It is also more tedious and confusing for the developer. A safer approach would be to create a Company class with attributes to store revenue, earnings, stock price, etc. and then instantiate Company objects for each company stock you want to evaluate. An array, list, or dictionary of Company objects could then be provided as input to a RatioCalculator class that could calculate the ratios that help to determine the investment value of each company.

Let's examine how we might write this simple program. We will focus on just the price/earnings ratio to keep things simple. The P/E ratio is simply the price per share of the stock divided by the earnings per share of the same stock. These values are provided in financial reports published regularly by publicly

traded companies. Figure 5.1 shows an example of how this program could be modeled with a class diagram.

**Figure 5.1. Class Diagram for Example Program**



### Looping with Objects

The program that follows is based upon the UML diagram in Figure 5.1.

```
class Company:  
  
    def __init__(self):  
  
        #The orders attribute contains a list that can be filled with Order objects  
        self.companyName = None  
        self.industrySector = None  
        self.price = None  
        self.earnings = None  
  
    #other attributes from company financial reports could be added as well  
  
    #If you wanted private class attributes, getters and setters could be added here  
    #In this example, the attributes are public, so getters and setters aren't necessary  
  
class RatioCalculator:  
  
    def calculatePERatio(price, earnings):  
  
        return price / earnings  
  
    #Other ratio calculation methods, such as PEG ratios, debt/equity ratios, etc., could go here  
  
#Code to execute the program  
  
#Create company objects you wish to analyze  
ford = Company()
```

```

google = Company()
dow = Company()

#set the attributes for each company. Only one example is provided below for brevity.
#this data would like come from a database or user interface, instead of being manually written
ford.companyName = "Ford Motor Company"
ford.industrySector = "Consumer Cyclical Industry"
ford.price = 9.90
ford.earnings = -0.19

#create an empty list to contain multiple Company objects
companies = []

#add the companies to the list
companies.append(ford)
companies.append(google)
companies.append(dow)

#instantiate a RatioCalculator
calculator = RatioCalculator()

#finally, loop over the companies and calculate the P/E ratio for each

#the for-in loop is used to extract individual Company objects from the companies list
#At each iteration, one Company object is placed into the company variable
for company in companies:

    #calculate the P/E ratio for the company at each iteration
    peRatio = calculator.calculatePERatio(company.price, company.earnings)

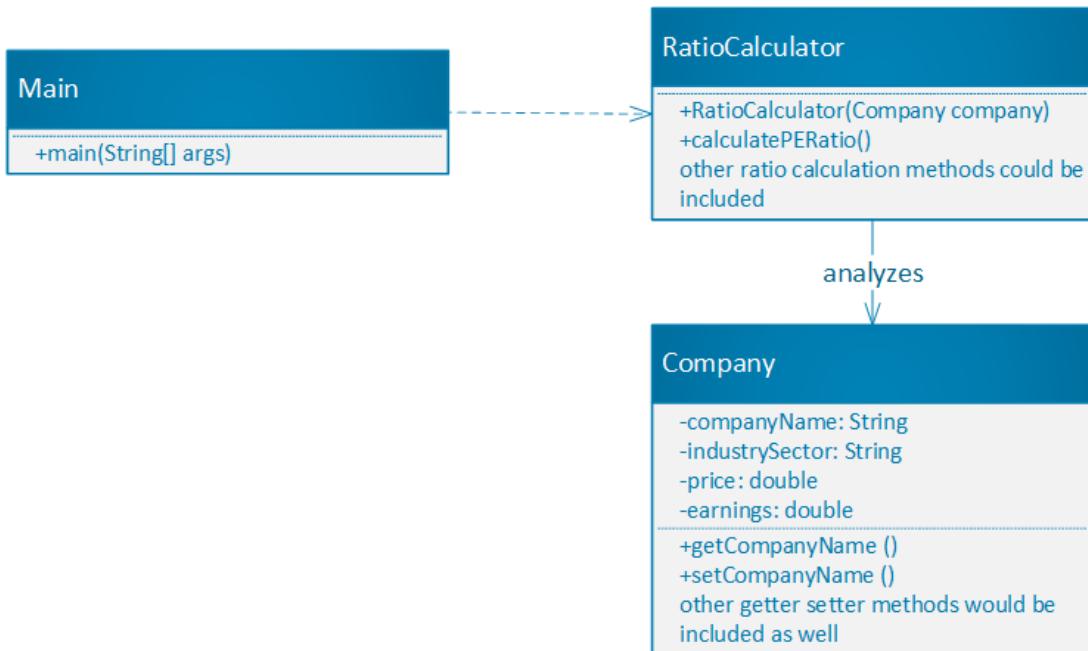
    #print the P/E ratio for each company as the loop iterates
    print("The P/E ratio for {} is {}".format(company.companyName, peRatio))

```

There are many different ways that this program could have been designed. The current example retrieves the price and earnings data from the Company object and then passes the price and earnings data to the calculatePERatio() method of the RatioCalculator object. Alternatively, the entire Company object could have been passed into the constructor of the RatioCalculator. Then, the calculatePERatio() method would not require the price and earnings to be passed in as arguments. Instead, the calculatePERatio() method could call the Company object stored within the RatioCalculator class to access the price and earnings data. Figure 5.2 shows what this might look like in a class diagram. Make note of the differences between Figure 5.1 and Figure 5.2.

This text doesn't cover design patterns and frameworks in detail. Other texts, such as those that focus on systems analysis and design or software design patterns, exist if you are interested in learning more about how to design interactions between classes. For now, just recognize that there are many ways to design the same program. Don't worry at the moment about which design is "best." Often, there isn't a clear "best" design, as designs possess different strengths and weaknesses. Often, design is about tradeoffs to meet the needs of a particular situation.

**Figure 5.2. Class Diagram for an Alternative Program**



### Looping with objects

The program that follows is based upon the class diagram in Figure 5.2. The **Company** class does not change in this alternate design. So, it is not included below. The **Company** class would look identical to the class created in the previous example.

```

class RatioCalculator:
    #The constructor is used as a setter to store a Company object in the company attribute
    def __init__(self, companyObj):
        #a company attribute is added in this design, which is passed in through the
        #constructor
        self.company = companyObj

    #No parameters are needed for this method in this design as the Company object is internal
    def calculatePERatio(self):
        return self.company.price / self.company.earnings

    #other ratio calculation methods, such as PEG ratios, debt/equity ratios, etc., could go here

#Execution of the program starts here

#Create multiple Company objects for each company you want to analyze as before
ford = Company()
google = Company()
dow = Company()
  
```

```

#Set the attributes for each company as before. Only one example is provided below.
ford.companyName = "Ford Motor Company"
ford.industrySector = "Consumer Cyclical Industry"
ford.price = 9.90
ford.earnings = -0.19

#In this design, we create a RatioCalculator for each Company
fordCalculator = RatioCalculator(ford)
googleCalculator = RatioCalculator(google)
dowCalculator = RatioCalculator(dow)

#In this design, the list stores RatioCalculators
calculators = []

#Add the companies to the ArrayList
calculators.append(fordCalculator)
calculators.append(googleCalculator)
calculators.append(dowCalculator)

#finally, loop over the Calculator objects and calculate the P/E ratio for each company
for calculator in calculators:

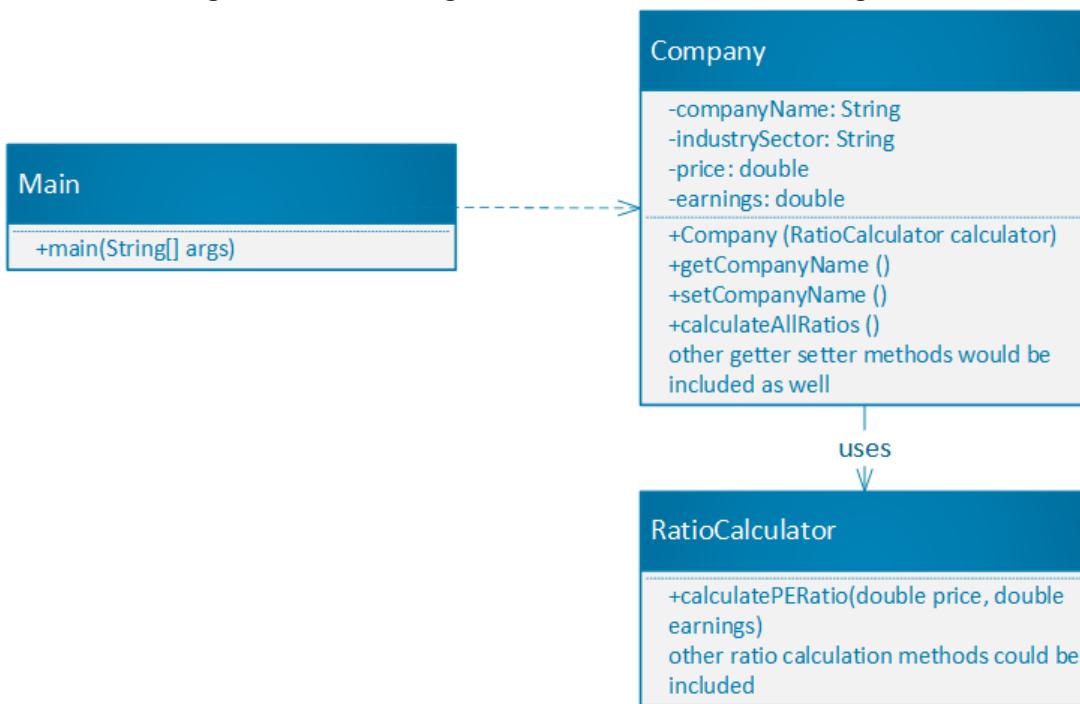
    #calculate the P/E ratio for the company stored in the Calculator object
    peRatio = calculator.calculatePERatio()

    print("The P/E ratio for {} is {}".format(calculator.company.companyName, peRatio))

```

Even more designs could be imagined, such as one in which a RatioCalculator is passed into a Company object and a calculateAllRatios() method is added to the Company object to calculate the ratios for the company. Figure 5.3 shows a class diagram with this design.

**Figure 5.3. Class Diagram for another Alternative Program**



The point of showing these different alternatives is to make you aware that there are multiple ways to design a program. For now, don't worry about the design. However, recognize that as a program's data requirements become more complex, the number of design options increases as well. The main takeaway of this example is that when data becomes complex, classes and objects can help you keep track of the data through the class attributes. Classes are customizable data structures.

## Recursion

Iterative loops are just one method of performing actions repeatedly. Another method, called recursion, deserves your attention. Recursion uses methods to create repetitive behavior. In **recursion**, a method calls itself until some condition is met. The circular call to itself creates a looping behavior. Without a stop condition, recursion would lead to an infinite loop in which a method calls itself endlessly. Recursion can be indispensable for some problems, but also exhibits some downfalls.

First, any time you call a method in a programming language, that method call gets put into something called the “call stack.” The call stack is a data structure that keeps track of the order in which methods were called in your program. It relates to flow control and was mentioned briefly in previous chapters. Because it is a data structure, the call stack consumes computer memory. By using a recursive method to repeat behaviors, the stack is updated each time the recursive method calls itself. In doing so, this increases the amount of memory your program consumes in the call stack. It can also create a stack that reaches the limit of information that can be stored in the call stack, leading the program to throw an Exception.

However, despite its potential downfalls, many developers love the simplicity and “beauty” of recursion. Tree data structures often rely on recursion to traverse all of the nodes in the tree. Some calculations are

hard and possibly impossible to perform without recursion. It is thus important to recognize it as a method for repeating behavior in programs.

Business analytics and machine learning relies heavily on matrices of data to perform calculations using linear algebra. Reading more about linear algebra will help you if you wish to start a career in business analytics. You don't have to be an expert in linear algebra to get started with a business analytics career, but some understanding will help you respect the algorithms you use. One important operation that is used frequently in many analytics algorithms is a matrix transpose. Conceptually, a matrix transpose involves rotating the values of a matrix along the main diagonal of the matrix. Figure 5.4 shows what rotating a matrix along its diagonal looks like. Notice that the values along the diagonal do not change, but the values on opposite sides of the diagonal switch places.

In terms of data structures, a matrix transpose is accomplished simply by changing the two index values for each value stored in the matrix. If you recall, a matrix is a multidimensional array (i.e., arrays stored within an array). So values in a matrix are represented by two separate indices. For example, the value 1 in the first matrix below is at position (0,0) in matrix notation or [0][0] as array indices, the value 4 is at position (0, 1) in matrix notation or [0][1] as array indices, and the value 5 is at position (1,2) in matrix notation or [1][2] as array indices. If that doesn't make sense, recall from Chapter 4 that the first index of a matrix represents the row position and the second index represents the column position of the value.

To transpose a matrix, you simply need to swap the row and column indices for each data point in the matrix. So the values at (0,0), (1,1), and (2,2) would all stay the same because the indices are the same for the row and column. That is why the values on the diagonals don't change. However, the value 4 at position (0,1) would change to position (1,0). Similarly, the value 5 at position (1,2) would move to (2,1) as can be seen in Figure 5.4.

**Figure 5.4: Transposing a Matrix is Equivalent to Rotating it Along the Main Diagonal**

$$\begin{bmatrix} 1 & 4 & 6 \\ 3 & 0 & 5 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 7 \\ 4 & 0 & 8 \\ 6 & 5 & 9 \end{bmatrix}$$

Although swapping indices is easy to do on paper, the logic requires some thought. Recursion can help us perform this transposition with relatively simple logic. Although the length of the comments in the code below makes the code appear long, the code itself is fairly short. This example brings together the use of multidimensional arrays, embedded if statements, and recursion. You can begin to see how flow control in conjunction with data structures can help you accomplish important operations.

### Recursive Methods

```

class MatrixTransposer:
    #the transpose() method accepts:
    # 1) the original matrix of values
    # 2) the transpose matrix that is slowly built as the transpose() method is recursively called
    # 3) the row index that needs to be changed in the current iteration of the recursive loop
    # 4) the column index that needs to be changed in the current iteration of the recursive loop

    def transpose(self, matrix, transposeMatrix, row, column):
        #We need to know the dimensions of the original matrix
        numberOfRows = len(matrix)
        numberOfColumns = len(matrix[0])

        #The first time the transpose() method is called, we have no transposeMatrix.
        #We need to create a transpose matrix during the first iteration. Notice that the size
        #of #the array (row size, column size) is swapped (column size, row size). The
        #number of #columns in the original matrix are now the number of rows in the
        #transpose matrix, #and the number of rows in the original matrix are now the number
        #of columns in the #transpose matrix. This is the first part of the swap.
        if transposeMatrix is None:
            transposeMatrix = np.zeros((numberOfColumns, numberOfRows), float)

        #This if statement contains the condition that stops the recursion. When the current
        #row is greater than the number of rows in the original matrix, the if statement ends
        #and a result is returned.
        if row < numberOfRows:
            #This if statement helps jump to the next row of the matrix when the current
            #column is greater than the number of columns in the original matrix.
            if column < numberOfColumns:
                #this line swaps the column and row indices to transpose the matrix
                transposeMatrix[column][row] = matrix[row][column]

                #This line is the recursive method call, calling to itself with
                #updated parameters. The column index is incremented by 1 to
                #move #to the next column in the row
                transpose(matrix, transposeMatrix, row, column+1)
            else:
                #This line is also a recursive method call, calling to itself with
                #updated parameters. The row index is incremented by 1 to move to
                #the first column of the next row in the matrix.
                transpose(matrix, transposeMatrix, row+1, 0)

        return transposeMatrix

#code to use the MatrixTransposer class
transposer = MatrixTransposer()

```

```
#the original matrix
matrix = np.array([ [1, 1, 1, 1],
                    [0, 1, 0, 1],
                    [1, 0, 1, 0] ], float)

#the initial call to the recursive transpose() method with starting parameters. The original matrix is
passed in. None is put in place of the transpose matrix parameter because we don't have a
transpose matrix at this point in the recursion. The row and column parameters are set to 0 so that
the recursion starts at row 0 column 0 of the matrix, (0, 0) or [0][0].
transposeMatrix = transposer.transpose(matrix, None, 0, 0)
```

If you understood the code in this example, great! You are making excellent strides in your understanding of algorithms and how they can be implemented with programming logic. If you didn't grasp everything from this example, that is okay. With exposure to programs and algorithms like this, your ability to understand them will increase. You must be patient with yourself and allow your mind to struggle with difficult concepts. In doing so, over time, you will excel at whatever endeavor you undertake.

This chapter has furthered your understanding of programming by demonstrating how flow control logic can be used in conjunction with data structures to provide more complex behavior.

## Chapter 5 Assignment

Please see the [Chapter Assignments Workbook](#) for the Chapter assignment.

# Chapter 6:

## Introduction to Data Analytics

Data analytics is one of the growing uses for programming. Many different industries (e.g., the banking industry and the medical industry) and organizational departments (e.g., accounting, marketing, and logistics departments) use data to improve products, automate decision making, or better inform human-led decisions. Thus, from a decision-making perspective, analytics can either replace or support decisions made by humans.

Ad hoc decisions (i.e., decisions that are made one time), such as whether to build a new manufacturing plant in a specific location, are generally human-led decisions that rely on analytics for support. It would be wasteful to write a completely automated analytics program for such ad hoc decisions, simply because these decisions only need to be made once. However, many operational decisions arise over and over again. These decisions can be automated through analytics and computer programming and result in huge savings to organizations over time. Ad hoc decisions rely on notebooks or other software to help analyze data to inform decisions. Long-term operational decisions often rely on notebooks for data exploration and model development. Then, the models are integrated into automated systems. The fundamentals of this chapter will help you learn basic analytics skills that are crucial to support either ad hoc decisions or exploration for automated operational decisions.

### Data Wrangling in Python

**Data wrangling** is the act of cleaning raw data so that it is prepared to be used in statistical models. Many data science professionals will tell you that upwards of 80% of their time is spent cleaning data. This section will identify how to get data from a file into a notebook, and introduce different ways to check the cleanliness of your data and clean the data.

Python is a common language for analytics applications because many useful analytics libraries have been built with functions to read, clean, visualize, and perform statistical analyses on data. One such library is the **pandas library**. Pandas comes pre-installed with Anaconda. To use pandas within a notebook, you must begin by importing the pandas library. Imports are added to the first cell of your notebook. To import pandas, simply add the following line to a cell:

```
import pandas as pd
```

### Reading data from csv and Excel files

Before you can clean data, you must read the data into your notebook and store the data in a variable for later use. The pandas library comes equipped with several methods to help you read data into your notebook. Spreadsheets can be imported into a notebook with ease using the pandas `read_csv()` or `read_excel()` methods. The `read_csv()` method is useful for reading comma and tab delimited .csv and .txt files. If you have a spreadsheet saved as .xlsx, you can use the `read_excel()` method. Both methods can accept a number of arguments to alter how the methods behave. Because you will not often need to utilize all possible arguments for these methods, you will use the named parameter technique in Python to

provide method arguments. See Chapter 2 if you don't recall the difference between named and ordered parameters. For a full list of the arguments the methods accept, you can visit the documentation pages at:

[https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_csv.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html) for `read_csv()`

[https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_excel.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_excel.html) for `read_excel()`

You don't need to know all of the arguments that you can pass to the `read_csv()` and `read_excel()` methods to get started. For the `read_csv()` method, passing the file location and the delimiter (e.g., comma or tab delimited) is enough to read in many .csv and .txt files. For the `read_excel()` method, passing the file location and sheet name (if multiple sheets are contained within the spreadsheet) is sufficient to read in many .xlsx files. The `read_csv()` and `read_excel()` methods **convert your data into a pandas data frame**. Recall from Chapter 4 that a data frame is a 2-dimensional data structure made of rows and columns.

By default, Excel saves .csv files as comma delimited. However, many .csv files are tab delimited. A delimiter is just a character (i.e., "," for comma delimited or "\t" for tab delimited) that separates the values in the rows of a spreadsheet. The spreadsheet hides these delimiters from you, so you may have never noticed them before. However, if you open a .csv file in a program like Notepad, you will see the commas or tabs between your data points.

```
import pandas as pd
```

**To read a csv file that is comma delimited**, you simply need to specify the file name and location as shown below. The comma delimiter is the default delimiter for the `read_csv()` method.

```
data = pd.read_csv("C:/path/to/file/filename.csv")
```

**To read a txt file that is NOT comma delimited (e.g., tab delimited)**, you need to specify the file name and location and the delimiter as shown below. The delimiter for tab delimited files is "\t". You pass the delimiter to a parameter called "sep" (i.e., separator). Notice that the method call below used a combination of ordered and named parameters.

```
data = pd.read_csv("C:/path/to/file/filename.txt", sep="\t")
```

**To read an Excel file with one sheet**, you simply need to specify the file name and location as shown below.

```
data = pd.read_excel("C:/path/to/file/filename.xlsx")
```

**To read an Excel file with multiple sheets**, you need to specify the file name and location and the name of the sheet as shown below. The sheet is specified through the `sheet_name` parameter, which is passed the name of the sheet you want to read into the program.

```
data = pd.read_excel("C:/path/to/file/filename.xlsx", sheet_name="Sheet2")
```

### Possible errors when reading Excel files

Recent versions of pandas require an additional library to support the reading of .xlsx files. Newer versions of the Excel reader in the pandas library only support .xls files. To read .xlsx files (i.e., the default file extension for Excel files), you must use another “engine” to read in the Excel file: openpyxl.

If you receive an error stating that a .xlsx file couldn’t be read, change the engine to openpyxl. First, you must ensure that you have openpyxl installed, which may be installed by default. If it isn’t, you can use the Anaconda Command Prompt to install openpyxl using the command: pip install openpyxl. With openpyxl installed, you simply need to specify a different “engine” for the read\_excel() method.

```
data = pd.read_excel("C:/path/to/file/filename.xlsx", engine="openpyxl")
```

### What is a column in a pandas data frame?

As suggested above, data frames store data in rows and columns. Columns represent “variables” within your data set. In data science lingo, they might also be called “features” and “labels”. Be careful not to confuse the term variable here with the way we have referred to variables in other chapters. When speaking in programming terms, a variable is a named storage bucket in memory that can contain objects or values. When speaking in data/statistical terms, a variable is a dimension or feature of your data set that holds a value that will likely remain unchanged. Let’s consider an example. Assume you have a data set with “variables” that represent employee productivity. These “variables” (i.e., columns/features) might be: “hours worked”, “products produced per hour”, “breaks taken”, “total minutes on break”, “employee satisfaction”, etc. Each row in the data frame would represent values for an employee. Thus, rows represent records in your data frame and columns represent features/variables. To avoid confusion, we will primarily refer to variables in a data frame as columns or features.

### Viewing data in data frames

After reading data into a variable, you may want to make sure that the data looks as you expect it to. However, you may not wish to see all of the data if the dataset is large. To **display the first few entries** in the data frame, you can call the **head()** method. To **display the last few entries** in the data frame, you can call the **tail()** method. You can use the **print()** method to print the entire dataset (depending on your notebook settings). You will rarely want to print out a large data set in its entirety.

The following code shows an example of displaying the first five entries in the data frame. If you were to run the cell with this code, the program would read in the csv file and then show the first five entries in the data frame as the output of the cell.

```
import pandas as pd  
data = pd.read_csv("C:/path/to/file/filename.csv")  
data.head()
```

The following code shows an example of displaying the last five entries in the data frame.

```
import pandas as pd  
data = pd.read_csv("C:/path/to/file/filename.csv")
```

```
data.tail()
```

The following code would print all of the code in the data frame. The print() display would not be formatted as nicely as it is with the head() and tail() methods.

```
import pandas as pd  
data = pd.read_csv("C:/path/to/file/filename.csv")  
print(data)
```

### **View subsets of data in a data frame**

At times, you do not want to view the entire dataset. You may want to focus on subsets of the data that meet some criterion. For example, you may want to only see data for individuals with high or low salaries. Or you may only want to focus on stocks with purchase prices above a certain threshold. Pandas provides many different ways to do this. We will explore one simple, yet effective filtering method. The filtering is done by setting a condition within hard brackets. In the code below, you will see some simple examples of filtering based on conditions. Later in this chapter, you will learn about lambda functions, which allow for more complex filtering using this method.

The following code will show you how to filter data to retrieve a subset of the data.

```
import pandas as pd  
  
data = pd.read_csv("C:/path/to/file/filename.csv")  
  
#the data subset is created with the condition: data.Salary > 70000  
#a new data frame with the filtered data is stored in a new variable named dataHighSalary  
dataHighSalary = data[data.Salary > 70000]  
  
#you can also add multiple conditions: (data.Salary > 70000) & (data.Age < 50)  
dataHighSalaryUnder50 = data[(data.Salary > 70000) & (data.Age < 50)]  
  
#multiple conditions with an or operator can also be accomplished with the pipe (i.e., | ) character  
  
#you can filter based on True conditions where True values are retained: data.isOnLeave  
dataEmployeesOnLeave = data[data.isOnLeave]  
  
#you can filter based on False conditions where False values are retained: ~data.isOnLeave  
#the tilde represents: not (i.e., False)  
dataEmployeesNotOnLeave = data[~data.isOnLeave]
```

At times, you may wish to **view data from only one column or a subset of columns**. You may want to scrutinize one or a few columns without having to see all of the columns. Large data sets can contain hundreds or thousands of features. To view one column, you can call the column by name or pass the

column name in bracket notation. To view a subset of columns, you will pass an array of column names in bracket notation. You can then call the head() or tail() methods to see data for the specified columns.

The following code will show how to view a single column of your data using bracket notation and the head() method. The code below assumes the data has a column named: Salary.

```
import pandas as pd  
data = pd.read_csv("C:/path/to/file/filename.csv")  
data['Salary'].head()
```

Alternatively, you could simply call to the name of the column as below (this method does not work if your column name includes spaces):

```
import pandas as pd  
data = pd.read_csv("C:/path/to/file/filename.csv")  
data.Salary.head()
```

The following code shows how to view a subset of columns of data using bracket notation and the head() method. The code below assumes the data has columns named Salary and Age. Notice the array containing Salary and Age.

```
import pandas as pd  
data = pd.read_csv("C:/path/to/file/filename.csv")  
data[ ['Salary', 'Age']].head()
```

If you are interested in exploring other more complex methods, research the filter(), query() and where() methods built into pandas. If you want to filter by the indices of the data frame, you can also research loc and iloc.

## ***Viewing the data types of columns***

You may also wish to ***review the data types*** of the data that you read into a data frame to make sure the data was converted to the correct data type. To do this, just call the dtypes attribute of the data frame.

The following code will show you the data types of the data stored in the columns of your data frame.

```
import pandas as pd  
data = pd.read_csv("C:/path/to/file/filename.csv")  
data.dtypes
```

The output of running the code above might look something like:

Name	object
Age	float64
Salary	object
dtype:	object

This output would tell you that the data frame contains three columns (i.e., features) named: Name, Age, and Salary. Name has an object data type, which usually means that it contains string data. Age has a float data type (i.e., decimal values). Salary has an object data type, meaning that the Salary column was read in as a string. If you were expecting the data in Salary to be converted to a float data type, you would want to use the head() or tail() methods to examine the data to figure out why the data was not converted to an int or float data type. For example, maybe the Salary data included a dollar sign or commas which are invalid characters for float data. In this case, the data would need to be cleaned before converting it to the proper data type.

At times, you may wish to **view data from only one column or a subset of columns**. You may want to scrutinize one or a few columns without having to see all of the columns. Large data sets can contain hundreds or thousands of features. To view one column, you can call the column by name or pass the column name in bracket notation. To view a subset of columns, you will pass an array of column names in bracket notation. You can then call the head() or tail() methods to see data for the specified columns.

The following code will show how to view a single column of your data using bracket notation and the head() method. The code below assumes the data has a column named: Salary.

```
import pandas as pd  
data = pd.read_csv("C:/path/to/file/filename.csv")  
data['Salary'].head()
```

Alternatively, you could simply call to the name of the column as below (this method does not work if your column name includes spaces):

```
import pandas as pd  
data = pd.read_csv("C:/path/to/file/filename.csv")  
data.Salary.head()
```

The following code shows how to view a subset of columns of data using bracket notation and the head() method. The code below assumes the data has columns named Salary and Age. Notice the array containing Salary and Age.

```
import pandas as pd  
data = pd.read_csv("C:/path/to/file/filename.csv")  
data[ ['Salary', 'Age' ] ].head()
```

## Finding and fixing incorrect data types and strange values

Although you may hope that your data is read into the program correctly, that isn't always the case.

**Sometimes data will be read into your program with an incorrect data type.** For example, if you had currency values, such as \$3,2123.00, the program might convert the currency data into a string data type instead of a float data type. Decimal values (i.e., the float data type) cannot contain special characters like \$ or commas. In such cases, you might first need to **strip out the extra characters** (e.g., \$ and

commas). Then, you can **convert the data to the correct data type**. To reiterate, the `dtype` attribute can be called on your data frame to identify which columns might possess an inappropriate data type.

Stripping out illegal characters can be accomplished in many different ways. Although beyond the scope of this text, you may want to learn about regular expressions. Regular expressions allow you to identify patterns in data and can be used to replace bad patterns, such as currency values starting with a dollar sign. Here, we will show you a simpler version of regular expressions, namely the `replace()` method. The `replace()` method allows you to replace a character with another character or with nothing. In the case of fixing currency data, we might wish to replace a \$ or commas with nothing. Once your data is properly formatted for the desired data type, you can use the `astype()` method to convert the data in a column to the desired data type.

You can also try to use the pandas `to_numeric()` method to force a column to convert from an object data type to a numeric data type. The `to_numeric()` function will throw an error if your column contains string values.

The code below shows how to replace a character in a string that prevents a column from being converted to a float data type. The code below assumes the data is in a column named: `Salary`. The code replaces the dollar sign with an empty string and then commas with an empty string.

```
import pandas as pd
data = pd.read_csv("C:/path/to/file/filename.csv")

#To make the change, you can store your changed data in a new variable
newSalaryData = data.Salary.replace("$", "") #replace $ with nothing for Salary column
data.Salary = newSalaryData #set the Salary column equal to the newly formatted data

#Alternatively, you can simply update the column in this fashion without the use of a new variable
data.Salary = data.Salary.replace(",","",) #replace the comma with an empty string
```

Further, you could replace the dollar sign and comma in one line with the following chain of commands:

```
import pandas as pd
data = pd.read_csv("C:/path/to/file/filename.csv")
newSalaryData = data.Salary.replace("$","",).replace(",","",)
data.Salary = newSalaryData
```

Once the data is properly formatted for the desired data type, you can use the `astype()` method as depicted below. If you don't format the data before trying to change the data type, you may receive an error stating that the column of data couldn't be converted to the desired data type.

```
data = data.astype({'Salary':'float'}) #change the data type of Salary to float
```

Alternatively, you could have used the `is_numeric()` method  
`data.Salary = pd.to_numeric(data.Salary)`

You may have noticed in the code above that temporary variables (i.e., newSalaryData) were created and then used to replace the data in columns, or the entire data frame was set equal to some altered version of the data frame (i.e., data = data.astype({'Salary':'float'})). Why are we overwriting the initial column or data frame with a new column or data frame? This practice is necessary within pandas because **pandas doesn't permanently save all changes you make to the data frame unless you explicitly tell it to do so**. This feature may seem strange, but it allows you to create different branches of your data with different values and formats to test your data in different ways without permanently altering the data in the data frame. If you don't want multiple versions of your data, you have to replace your old data with the altered data by explicitly setting the column or entire data frame equal to the altered data. In later examples, you will see how to accomplish the same task with a parameter called "inplace".

At times, the reason that your data was marked with the incorrect data type is not readily apparent by examining just the head() or tail() of the data. Your data may appear at first glance to be formatted correctly, but is read in as an object data type. **At times, it may only be one or two strange values among thousands or even millions of rows that causes data to read in with an incorrect data type.** The head() and tail() methods may not help you find these issues.

For example, a couple of numbers may have been spelled out in letters (i.e., forty and six instead of 40 and 6). Similarly, you might have strange data in a few rows (i.e., @KLSJd in a Salary column). One way to identify these strange values is to print out the unique values within a column using the unique() method. This trick won't work if every value in a column contains a different value, but many columns contain a small set of repeated values. If you look through the unique values and find a strange value, you can identify which row has the offending value by using the bracket notation as depicted in the example below. You can then replace the strange value or delete the row with the strange value. In the upcoming section about lambda functions, a slightly more advanced way to find issues with data types is presented.

To list out the unique values within a column, you can call the unique() method in pandas for the particular column you wish to examine.

```
import pandas as pd  
data = pd.read_csv("C:/path/to/file/filename.csv")  
data.Age.unique() #this method will show all of the unique values in the Age column
```

Assume the output of the unique() method returned the result below. It is clear that the values 'forty' and '8fW' are preventing the column from being converted to the correct data type.

```
array(['0', '80', '31', '9', '93', nan, '72', '19', '12', '61', '77',  
      '39', '22', '67', 'forty', '71', '24', '52', '73', '87', '8fW',  
      '32', '333', '18', '51', '42', '56', '82', '78', '66', '58', '88',  
      '65', '49', '55', '62', '27', '10', '60', '40', '11', '-5', '75',  
      '50'], dtype=object)
```

Once you know the offending value(s), you can use bracket notation with a test for equality to identify which row or rows in the data frame have the strange value.

```
import pandas as pd  
import numpy as np
```

```
data = pd.read_csv("C:/path/to/file/filename.csv")
data[data.Age=="forty"] #retrieves all rows where Age is equal to the word 'forty'
```

The code above would print out the rows in the data frame that had a value for Age set to "forty". Next, you must decide how to remedy the strange values. For example, you might use the replace() method to replace "forty" with 40. If the value were more obscure, such as "8fW", you could choose to delete the row or replace the value with a null value. Null values are created by calling the nan attribute of numpy: np.nan. If you see nan or NaN in your data frame, that simply means the value is null (i.e., empty). To replace a strange value with a null value, use the commands below.

```
data.Age = data.Age.replace("forty","40") #convert the word string to a number version of the string
data.Age = data.Age.replace("8fW", np.nan) #convert the value to a null value
```

Once you have corrected strange values, you can change the data type of the column as shown earlier. However, some data types, such as int won't allow you to convert data with null values. You may need to replace null values before converting to certain data types. Replacing null values is discussed in later sections of this chapter.

## Identifying and correcting outliers

In addition to finding and replacing strange values that prevent you from converting columns to the appropriate data type (e.g., text in number columns, numbers in a name column, etc.), you will also want to look for extreme values. Extremely high or low values are called **outliers**. For example, assume that you possess salary data and the majority of your data exhibits salaries between \$30,000 and \$90,000. As you look at the data closely, you notice that one person has a salary of \$123,456,789 and another person has a salary of \$3. These are clearly strange data values for the particular data set.

**The first question you must ask yourself is whether these values are even legitimate responses or just data entry errors.** That isn't always easy to determine, and often requires some guesswork. The large number above is extreme for the particular data set and seems to be a sequence of numbers from 1 to 9, possibly caused by a data entry error. The \$3 salary also seems like it might be due to a data entry error as it is an illogical value for someone's salary. If you have reason to suspect that extreme values are the result of a data entry error, you might choose to set the values to null, such as using the replace() method with np.nan. You can then decide how to handle the missing data points (the topic of handling missing values is discussed later).

**Other outliers won't be clear data entry mistakes.** For example, you might have a salary of \$10,000 or a salary of \$195,000 in your dataset. These might be on the extreme for your particular dataset, but are within the realm of being real data points. In these cases, you must decide whether you want to drop the row with the extreme value or not. Outliers can affect the results of certain statistical analyses. For example, outliers can affect the value of the mean of your data. The **mean** is a measure of central tendency that is calculated by adding the values of all of the data points and then dividing the resulting value by the number of data points in the data set. Take for instance a data set with the following data points:

10, 11, 9, 12, 10, 12, 8, 10, 11, 154

The mean would be 247/10 data points or 24.7. Does 24.7 really seem like a representative value for this data set? If you remove the outlier (i.e., 154), the mean is 93/9 data points or 10.33, which seems like a more reasonable representation of the values in the data set. Many statistical methods rely on mean calculations for analysis and predictions. ***Some of these statistical methods can produce incorrect results if outliers inflate or deflate the mean.*** If your statistical methods are sensitive to outliers, such as some methods that rely on mean values, you may want to run the statistical analysis with and without the outliers included. If the results for both analyses are equivalent, it may be okay to retain the outliers. If the outliers alter the results of the analysis, you may want to drop the rows with outliers. Thankfully, not all statistical analyses are sensitive to outliers. Many robust statistical methods and machine learning models exist that allow you to worry less about the effect that outliers might have on results. As you learn about statistics and machine learning, pay attention to the assumptions being made within the statistical methods, such as their sensitivity to outliers.

A simple and rudimentary way to look for outliers is to sort your data in ascending and then descending order with the sort\_values() method and then use the head() method to examine the highest and lowest set of values. With this method, deciding what counts as extreme becomes a judgment call based only on individual perception. Still, you can get an early feel for whether outliers exist by sorting your data.

To sort data to find extreme values, you can call the sort\_values() method in pandas.

```
import pandas as pd
data = pd.read_csv("C:/path/to/file/filename.csv")
#this method chain will sort by Age in ascending order and return the 10 smallest values
#the "by" parameter allows you to set the column you want to sort by
data.sort_values(by="Age", ascending=True).head(10)
```

Assume the output of the method call returned the age -5 and 0 for a dataset of employees. These would clearly be incorrect data values. So you might then choose to replace the values with a null value. The code below could do that for you. In the next section, you will be introduced to lambda functions. With a lambda function, you could replace values with a null value that were less than a specific value (i.e., values < 16) with a single line of code to make replacement simpler.

```
data.Age = data.Age.replace(-5,np.nan)
data.Age = data.Age.replace(0,np.nan)
```

```
#this method chain will sort by Salary in descending order and return the 10 largest values
data.sort_values(by="Salary", ascending=False).head(10)
```

Assume the output of the method call returned the salary \$123,456,789 for a dataset of employees. As discussed above, this value could reasonably be considered a data entry error. So you might then choose to replace the value with a null value. The code below could do that for you.

```
data.Salary = data.Salary.replace(123456789, np.nan)
```

There are ***more sophisticated statistical analyses for identifying outliers than sorting your data*** and manually looking for extreme values. A simple, yet slightly better way of finding outliers is through the use

of a box plot (a.k.a., box and whiskers plot). A **box plot** is a graphical tool that allows you to see whether your data contains outliers based on the interquartile range. The **interquartile range** (IQR) measures the middle 50% of the data to identify how the data is spread. Data that falls too far outside of that middle 50% is considered an outlier. The IQR is calculated by first finding the **median** of the entire data set (i.e., the value in the middle of the data set when the data is sorted from smallest to largest). The data above and below the median is then grouped and the median of the upper and lower halves of the data are calculated. The median of the lower half of the data represents the first quartile, the median of the entire data set represents the 2nd quartile, and the median of the upper half of the data represents the third quartile. The interquartile range is calculated by subtracting the median of the upper half (i.e., the third quartile) from the median of the lower half (i.e., the first quartile). Let's take the example data set provided earlier with a slightly less extreme outlier (i.e., 16 instead of 154). First, we must arrange the data from smallest to largest.

8, 9, 10, 10, 10, 11, 11, 12, 12, 16

Because the data set has an even number of data points, the median (i.e., the second quartile) is the mean of the middle two numbers (i.e., the mean of 10 and 11:  $21/2 = 10.5$ ). You would then divide the data points in half and find the median of each half. The median values of each half of the data are bolded in the data below.

(8, 9, **10**, 10, 10) (11, 11, **12**, 12, 16)

Based on the analysis, the first quartile is 10, the second quartile is 10.5 as previously calculated, and the third quartile is 12. The interquartile range would then be the third quartile minus the first quartile (i.e.,  $12-10 = 2$ ). Also notice that the data is divided into four sections with the first, second, and third quartiles acting as dividers between the four sections (i.e., 8 & 9, 10 & 10, 11 & 11, and 12 & 16). To identify an acceptable range of values that lie outside of the middle 50% (i.e., the values between the first and third quartiles), box plots have **whiskers**. Whiskers are calculated by multiplying the interquartile range by 1.5 (e.g.,  $2 * 1.5 = 3$ ) and subtracting that number from the 1st quartile to get the lower whisker (i.e.,  $10-3 = 7$ ) and adding that number to the 3rd quartile to get the upper whisker (i.e.,  $12+3=15$ ). **Values that are less than the lower whisker or greater than the upper whisker are identified as outliers**. Based upon the analysis, 16 would be considered an outlier for the data set as it is greater than the upper whisker (i.e., 15). Again, whether you need to worry about outliers depends on the type of statistical method you plan to run on the data.

Thankfully, you don't need to make these calculations by hand or even numerically. Visual box plots can be created in pandas by calling the **boxplot()** method on the data you wish to examine. Figure 6.1 shows a box plot of the data just analyzed. As you can see from the plot, the value of 16 lies outside of the whiskers and would be considered an outlier. The box plot also shows that the values are not equally distributed. For example, between the first and second quartile, the values range from 10 to 10.5. However, from the second quartile to the third quartile, the values range from 10.5 to 12. The whiskers are also unequal. The lower whiskers reach to 8, but the upper whiskers only reach to 12 because of how the data is distributed.

If you were running a statistical model that was sensitive to outliers, you might run the model with the outlier included and then again with the outlier removed. Depending on how extreme outliers are and how the data is distributed, **outliers may not affect statistical results enough to justify removing them**.

However, they certainly can affect results. It is best to test the effect of outliers on the results of statistical models, particularly when a particular statistical model is known to be sensitive to outliers.

```
#the line below creates a new dataframe with a named column.  
data = pd.DataFrame([8,9,10,10,10,11,11,12,12,16], columns=["Age"])
```

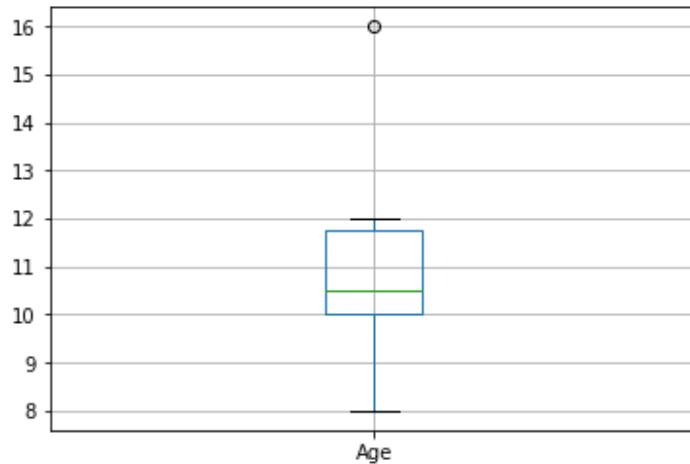
```
data.boxplot() #produces a boxplot from the data
```

If your data has multiple columns and you only want a box plot of a specific column, you would simply pass the column name into the boxplot() method. You can also pass an array of columns if you want box plots for a few columns, but not all columns.

```
data.boxplot("Age") #produces a boxplot from the data in the Age column
```

```
data.boxplot(["Age","Salary"]) #produces boxplots from data in the Age and Salary columns
```

**Figure 6.1: A Box Plot Showing an Outlier**



Another more sophisticated way to identify outliers is through z-scores. A box plot shows the distance of data points from the median based on the raw scores (i.e., actual values) of the data column. For example, the outlier value of 16 shown in Figure 6.1 is the raw score (i.e., 16) from the data set presented earlier, which is 5.5 units away from the median of 10.5. The axis of the box plot in Figure 6.1 displays 8-16, which represent the range of raw scores found in the data set. Conversely, a **z-score** shows standardized distance between a data point and the mean (typically converted to a mean of 0). Z-scores are created by converting raw scores to standardized scores by taking a data point, subtracting it from the mean and then dividing that value by the standard deviation. If you are not familiar with the **standard deviation**, it is another measure like quartiles of the spread of your data. The formula for the standard deviation (i.e., $s$ ) is:

$$S = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n-1}}$$

**Where:**

- $\Sigma$  is a mathematical symbol that tells you to loop over the data points and perform the  $(x_i - \bar{x})^2$  calculation for each data point.
- $x_i$  is the  $i^{th}$  data point for a particular column in the dataset that is looped over. “ $i$ ” is like the loop counter discussed in earlier chapters.
- $\bar{x}$  is the mean for the set of data points for the column in the dataset.
- $n$  is the number of data points for the column in the dataset.

Thankfully, pandas and other Python libraries help you calculate the standard deviation easily through the `std()` method of a pandas dataframe.

```
#the line below creates a new dataframe with a named column.
data = pd.DataFrame([8,9,10,10,10,11,11,12,12,16], columns=["Age"])
data.std() #calculate the standard deviation of the Age column
```

Once you know the standard deviation, you can calculate the z-score of a particular data point. The formula for calculating a z-score is:

$$Z - score = (x - \bar{x}) / s$$

**Where:**

- $x$  is the value of a data point from a variable/column in the dataset.
- $\bar{x}$  is the mean for the set of data points from the variable/column in the dataset.
- $s$  is the standard deviation for the set of data points from the variable/column in the dataset.

**So why do z-scores rely on a standardized measure of distance instead of raw scores?** Let's explore an example. Although raw scores are okay to use in some instances, Figure 6.2 depicts an issue with raw scores. If you examine the plot in Figure 6.2, you will notice that the box plot of salaries looks the way you might expect. However, the age box plot is just a green line at the bottom of the graph. The raw scores for a variable like age might range from 1-90, but the raw scores for salary might range from \$25,000 to \$200,000. The magnitudes of the scores (i.e., 1-90 vs. 25,000-200,000) are very different for age and salary. Figure 6.2 is simply squishing the age box plot because of the drastic difference in the magnitudes of raw scores between age and salary.

**Figure 6.2: Box Plots of Age and Salary**

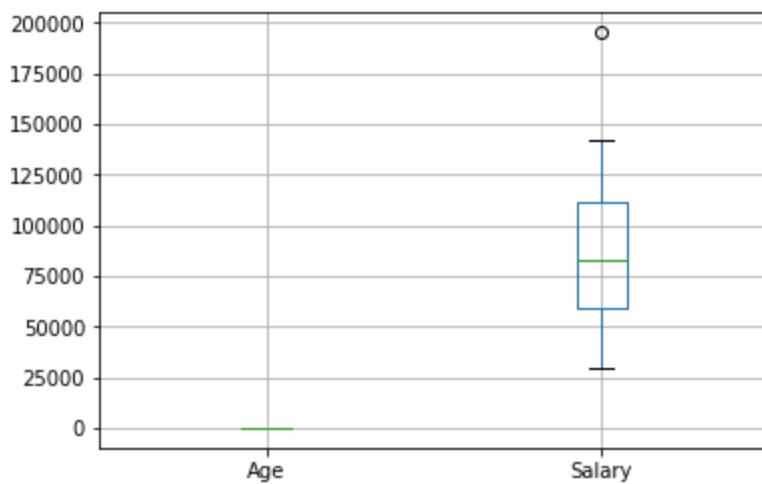
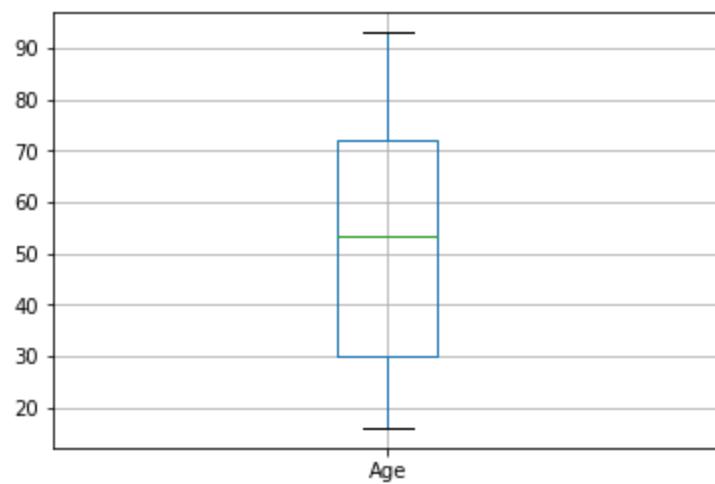


Figure 6.3 shows the age box plot from Figure 6.2 by itself. When not compared to salary, the age box plot looks like it should. In certain statistical methods, comparing raw scores can create issues because the raw scores give extra weight to variables with larger magnitudes (e.g., Salary) than to variables with smaller magnitudes (e.g., Age).

By converting all of your raw variable scores (i.e., column scores) to z-scores, z-scores for each variable typically lie in a range from -4 to +4 (a majority of which are often between -2 and 2). That means variables with large raw score magnitudes (e.g., Salary) are comparable to variables with small raw score magnitudes (e.g., Age). Both variables would possess standardized scores of a similar magnitude, making certain types of comparisons simpler. When using z-scores, scores above 3 or below -3 are typically perceived as outliers.

**Figure 6.3: A Box Plot of Age By Itself**



To convert your raw scores to z-scores in Python, you can create a new column in your data frame for each column you wish to convert to z-scores. For example, if you had a column named Age, you could

create a new column called AgeZScores. You would then fill this new column with the z-scores by using the z-score formula. In the example below, the value 154 is added to the dataset to better demonstrate outlier z-scores and how to drop extreme values. If you decide to drop a value, you will want to calculate the z-scores again. The z-scores change when you remove a value because the mean and standard deviation of the data change. In the example below, 154 could be designated as an outlier, but 16 would not be an outlier.

```
#the line below creates a new dataframe with a named column.  
data = pd.DataFrame([8,9,10,10,10,11,11,12,12,16,154], columns=["Age"])
```

the line below calculates the z-scores for each data point stored in Age and creates an entry in the AgeZScores column for that data point

```
data["AgeZScores"] = (data["Age"] - data["Age"].mean()) / data["Age"].std()  
  
data["AgeZScores"].sort_values(ascending=False)
```

the line above would output

```
10  3.01  
9   -0.18  
8   -0.28  
7   -0.28  
6   -0.30  
5   -0.30  
4   -0.32  
3   -0.32  
2   -0.32  
1   -0.35  
0   -0.37
```

The element at index 10 (i.e., value 154) is 3.01 standard deviations from the mean, which is greater than 3, suggesting it might be an outlier that you want to use carefully. All other data points have values between -3 and 3. The line of code below could be used to drop the outlier data point.

In the line of code below, the drop() method with inplace=True makes the change permanent. Without inplace=True, the dataset would still contain the row with value 154. The line `data[data["AgeZScores"]>3].index` gets the index of all data points whose z-score is greater than 3. The drop() method drops all of the indices that meet the criterion (i.e., scores> 3).

```
data.drop(data[data["AgeZScores"]>3].index, inplace=True)
```

You could then run the z-score calculations again with the outlier removed

```
data["AgeZScores"] = (data["Age"] - data["Age"].mean()) / data["Age"].std()
```

## Working with lambda functions

As you have seen through the examples in this chapter, the pandas library provides the ability to change rows within a dataset with a single line of code without using loops. The looping happens behind the scenes and is handled by the methods in the pandas library. Although many changes can be made to a dataset by using simple methods, such as the replace() and drop() methods, at times you will need to perform more complex cleaning logic on your data. Rather than writing a method and calling the method for complex calculations, many data scientists prefer to rely on lambda functions.

**Lambda functions** are methods with no name that can be passed as arguments to another method instead of being called on an object. That is, a lambda function is a method that can be passed as input to another method. In pandas, lambda functions are often passed as arguments to the apply() method of a data frame. The apply() method allows calculations to be applied to all data points in rows or columns of the data frame.

In Python, a lambda function is created by referencing the "lambda" keyword. Next, the parameter(s) of the lambda function are specified and a colon is used to designate the start of the lambda function body/block. In the code example below, the "x" represents the lambda function parameter. In the example below, the body of the lambda function is left empty.

```
data.apply(lambda x: ...)
```

You can call the apply method on the entire data frame, as in the code above, or on a specific column, such as Age in the example below.

```
data.Age.apply(lambda x: ...)
```

The body of the lambda function can be used to perform calculations or change specific rows or columns based on some condition (i.e., using if statements), much like the methods you have seen to this point. The syntax can be slightly different for lambda functions though. For example, if statements don't follow the typical if statement syntax. In lambda functions, the action to perform if the condition is true is the first element in the if statement. Then, the if condition itself is provided. Finally, an else statement is provided at the end. For example:

```
data.Age.apply(lambda age: np.nan if age < 16 or age > 100 else age)
```

In the code above, the lambda function inside the apply() method takes the age for each row in the Age column and checks to see if the age is less than 16 or greater than 100. If the age is less than 16 or greater than 100, the value for the row is set to null by calling `np.nan`. Otherwise, the age for the row is set to age (i.e., the current age value is retained). Again, most methods in pandas are going to perform the specified action for all rows/columns. The looping happens behind the scenes. When you use the apply(), replace(), drop() and other such methods, the request actions are performed on all requested rows/columns.

One concern with lambda functions is that they are not always easy to understand. So, you must decide if members of your team are comfortable with lambda functions. If they are not familiar with lambda functions, you might not want to use them. Shortcuts like lambda functions simplify work for those who

know how to use them, but can confuse those who do not. Unfortunately, manipulating data without lambda functions is tedious and requires far more lines of code. So, it may be best to train a team to use lambda functions if they do not know how to use them.

### **Lambda function to identify data type issues**

In a previous section, you learned how to fix issues with data imports by setting columns with the wrong data types, such as the astype() method and the to\_numeric() method. However, those methods won't always work correctly when strings are mistakenly embedded into a column intended to be numeric. Previously, the unique() method was presented as a simple way to look for string data mixed in with numeric data. However, the unique() method doesn't work well when you have many unique values. Lambda functions can help you identify strings amongst numeric data with greater ease.

To do this, we will identify all of the values in a column that are strings where there should be numeric values. We will store these rows in a separate data frame. You can filter out data in this way by specifying a subset of data within hard brackets. In the example below, the Salary column from the data frame stored in data is filtered with the lambda function inside the hard brackets. The filter looks for values that are not float values by calling a custom function named isFloat(). The isFloat() method is depicted below as well. By adding the "not" key word before the method call, only the values that evaluate to False will be filtered into the new data frame. In this case, the new data frame is stored in a variable named df. The actual lambda function is:

```
def isFloat(value):
    if value is None:
        return False
    try:
        float(value) #a built-in Python function that checks if a value is a float
        return True
    except:
        return False

data.Salary.apply(lambda value: not isFloat(value))
```

In the lambda function, the values of the Salary column are passed through the custom isFloat() method. The isFloat() method accepts a value and determines the value is a float or some other value. If the value is a float, the method returns True. Otherwise, the method returns false. In the code below, the lambda function is embedded inside the hard brackets to filter out all rows with Salary values that aren't floats (i.e., the problematic data). The variable df will now contain a data frame with a subset of the data. You can then use the head() method to identify the unique string values, which should be limited in number. You can then use the replace() methods to set string values to null or change textual numbers to actual numbers. If you loop over the newly created df data frame, you can automate this instead of manually calling replace() on each incorrect value.

```
df = data[data.Salary.apply(lambda value: not isFloat(value))]
df.head()
```

Alternatively, you can check if a value is not numeric, which is a built-in Python method. This approach saves you the extra step of creating your own customer function. However, you may not always have a

built-in function that performs the filtering that you desire. In such cases, you must create your own customer filtering method. You are now prepared for either situation.

```
df = data[data.Salary.apply(lambda value: not value.isnumeric())]
df.head()
```

Instead of trying to identify strings, you can also use lambda functions to only select numeric values. You simply need to reverse the logic of the previous lambda functions.

```
data = data[data.Salary.apply(lambda value: value.isnumeric())]
data.head()
or
data = data[data.Salary.apply(lambda value: isFloat(value))]
data.head()
```

Lambda functions provide the means to perform tasks quickly and simply that could require tedious and copious amounts of data sifting. For example, the unique() method could require you to look over hundreds or thousands of unique values. Use lambda functions when you can.

## **Identifying and replacing missing values**

Many datasets from statistical textbooks are clean and tidy. They don't have odd values, nor do they have missing values. Unfortunately in practice, you will rarely find such pristine data sets. Data sets in the real world contain missing and odd values. You've just learned some simple strategies to deal with strange values and outliers. You will also need to deal with missing values. Finding missing values can be done with the filtering method presented earlier in the chapter. You can filter by individual columns or by multiple columns:

```
dataNull = data[data.Salary.isnull()]
dataNull = data[(data.Salary.isnull()) | (data.Age.isnull())]
```

There are many ways to handle missing values. If a particular column has very few complete values, you might **drop the entire column**. If a row has very few complete values, you might **drop the entire row**. If a row only has a few missing values, you might choose to **replace the missing value** with a reasonable value. Filling missing values with a new value is called **data imputation**. But what is a reasonable value for imputation? The answer to that question is complicated and could be the topic of an entire course. We will present a few of the simplest forms of data imputation and briefly describe some other more advanced forms of imputation.

Simple forms of imputation include: filling missing values with a **default value** that you choose (e.g., fill missing ages with 20), filling missing values with the **mean or median value** of the column (e.g., fill missing ages with the mean or median age of the data set), filling missing values with a **nearby value or the average of nearby values** (useful for sequential data where nearby data points are related), or filling missing values with a **random value** within the range of possible values (e.g., fill missing ages with a random value between the minimum and maximum ages of the dataset).

Imputing missing values can be accomplished with the pandas fillna() or lambda functions and the apply() method. The fillna() method is limited in how it assigns values. For example, it is difficult to assign missing

values with a random value using the `fillna()` method. However, you can use the `apply()` method and a lambda function to perform such actions. Let's see some examples of replacing missing data using the `fillna()` and `apply()` methods.

To impute data with pandas, you can use the `fillna()` method on the data frame or a column of the data frame.

```
import pandas as pd  
data = pd.read_csv("C:/path/to/file/filename.csv")
```

The `fillna()` method can be used in a variety of ways. If you want the changes to be permanent, you must use `set inplace=True` or set the data frame or column equal to the changed data frame or column.

For details about all of the ways you can use the `fillna()` method, visit:

<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html>

For example, the `ffill` (i.e., forward fill) method will fill missing values with the value in the series that comes just before the missing value.

```
data.Age = data.Age.fillna(method="ffill")
```

The `bfill` (i.e., backward fill) method will fill missing values with the value in the series that comes just after the missing value.

```
data.Age.fillna(method="bfill", inplace=True)
```

You can also provide a value to the `fillna()` method, such as a default value or the mean/median of the column.

```
data.Age = data.Age.fillna(value=22)  
data.Age.fillna(value=data.Age.mean(), inplace=True)
```

You can also use lambda functions to help with data imputation.

To impute data with pandas, you can also use the `apply()` method with lambda functions on the data frame or a column of the data frame.

```
import pandas as pd  
import random  
data = pd.read_csv("C:/path/to/file/filename.csv")
```

Assume you wanted to impute missing values with a random value from the range of possible values. This could be accomplished with a lambda function. The code below makes use of the random library in Python to generate a random number, which is why the random library is imported above. The code also uses the `min()` and `max()` methods to find the minimum and maximum values of the Age column.

```
data.Age = data.Age.apply(lambda age: random.randrange(data.Age.min(), data.Age.max()) if  
pd.isnull(age) else age)
```

In the line of code above, the lambda function creates a parameter called age, which gets filled with each age value in the Age column. If the age value is null (i.e., missing), the random.randrange() method is called to change the age value for that row to a random age between the min and max age. The randrange() method accepts two arguments, namely the lower and upper values of the range. This is similar to the RANDBETWEEN() function in Excel. If the age isn't null, the age is set to its current value (i.e., nothing changes).

More complicated imputation methods include the use of linear regression or other statistical methods to identify a value for a missing data based on the values in other columns. Interpolation can also be used to fill missing values. For example, linear interpolation simply fills in a missing value with the value that falls on a line that connects the data point before and after the missing value. Linear interpolation or other more complex forms of interpolation can be useful for imputing missing values in time-series data (i.e., data that is measured over multiple time points) like stock price data. Pandas has a built-in interpolation method that can assist you in replacing missing values via interpolation. Using linear interpolation only requires one line of code. Assume you had a data frame full of stock prices for the company Meta, but some values in the time series were missing. You would simply need to write the following to fill in missing values for the open price of the stock. Of course, this example assumes the stock data is stored in a variable named stockData, which contains an attribute named openPrice.

```
stockData.openPrice = stockData.openPrice.interpolate()
```

If you want to learn other more complex forms of interpolation and how to execute them, refer to the pandas documentation here:

<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.interpolate.html>

## Collecting Stock Data with Python and APIs

Explaining and predicting financial data, such as stock prices, is an important business analytics objective for many organizations. Financial technology is expected to be a multi-trillion dollar industry. For this reason, this section will demonstrate how to retrieve stock data. In a later chapter, you will learn how to extract data from the annual reports of publicly traded companies.

To get price and volume data about a specific stock, you will need to know the stock ticker symbol. A **stock ticker symbol** is an abbreviated alias for the name of a company that publicly trades its stock. If you don't know what the stock ticker symbol is for a specific company, you can look it up on several different websites, such as: <https://finance.yahoo.com/lookup/>. If you type the company name, these sites will provide you with the ticker symbol.

## Using third-party data collection libraries

There are many libraries and application programming interfaces (APIs) that you can use to collect stock data. For example, the pandas library has an extension called **pandas-datareader** that allows you to quickly and easily collect stock data from multiple sources. The simplest approach within the pandas-datareader library is through their Yahoo Finance API integration in pandas-datareader. If you would like to look into the pandas-datareader library, see the documentation at: <https://pandas-datareader.readthedocs.io/en/latest/>.

This library, or others like it (e.g., yfinance at <https://pypi.org/project/yfinance/>), can be the simplest way to access data from sources like Yahoo Finance. However, these libraries aren't always regularly updated. When Yahoo Finance changes their API, if the libraries aren't updated quickly, your code won't work. For this reason, we will also show you how to access a stock data API directly. First, let's examine how to use pandas-datareader.

**To install pandas-datareader** in Anaconda on Windows, you will need to open the **Anaconda Prompt** instead of the Anaconda Navigator. Mac/Linux users will need to use the Terminal to install the library instead of using Anaconda Prompt. If you aren't using Anaconda, you can also install pandas-datareader with the pip install command in a Linux/Unix terminal or Windows Command Prompt if you have installed pip on your machine. The pip tool allows you to download a variety of Python libraries.

After opening Anaconda Prompt on Windows or the terminal on Mac/Linux, you will type the following command and hit Enter:

**pip install pandas-datareader**

You may have a different version of pip installed on your machine. In which case, you might need to try:  
**pip3 install pandas-datareader**

Alternatively, you can type the following Anaconda command and hit Enter:

**conda install -c anaconda pandas-datareader**

These commands will install pandas-datareader onto your respective computer so that you can use it within your notebooks. You can now begin to access stock data through your notebooks. If you experience an error message while using pandas-datareader, you may want to update/upgrade the pandas-datareader package.

For Windows, Mac, or Linux users, this can be done with the following command:

**pip install pandas-datareader --upgrade**

or if you have a different version of pip:

**pip3 install pandas-datareader --upgrade**

It can also be achieved with the Anaconda command:

**conda update -c anaconda pandas-datareader**

Again, unless a library is updated regularly, even updating the library with pip or conda commands will not fix error messages you might receive. You are at the mercy of the maintainers of a code library when you choose to use it in a program.

Other libraries like pandas-datareader, such as yfinance, can be installed and updated just as easily. Once installed on your machine, you can use the library within a notebook by importing it:

```
import pandas_datareader as reader  
import yfinance as yf
```

These libraries make accessing data simple. For example, the following line of code will collect data through the Yahoo Finance API for the specified ticker and time range and store the data in a pandas data frame:

```
#the following line reads data from Yahoo Finance to get Apple (i.e., APPL) price data from 2009-2019  
appleData = reader.DataReader("APPL", start="2009-1-1", end="2019-12-31", data_source="yahoo")
```

The data frame created by DataReader() will include several columns: Date, High, Low, Open, Close, Volume, and Adj Close (i.e., adjusted close).

## **Using a stock data API directly**

To gain more control over your access to stock data, it can be advantageous to access an API directly instead of relying on a third-party library. Many stock data API's exist. We will now examine the Alpha Vantage API, as it was created for student learning and academic research. The company has since developed commercial tools too.

**An application programming interface (API) is the interface through which one program communicates with another.** In the case of Alpha Vantage, the API allows your program to ask the Alpha Vantage system for stock pricing data for a particular company. APIs are often protected through authentication mechanisms, such as an API key that acts similar to a password. Your program will use the API key as a password to access the data from the system providing it (e.g., Alpha Vantage).

You can get an **API key for free from Alpha Vantage** by visiting:  
<https://www.alphavantage.co/support/#api-key>.

This free key has limits on the number of times you can request data per minute and per day. Your professor may be able to get you a key with unlimited access because Alpha Vantage has a mission to support educational access to stock data for research and teaching.

APIs have endpoints, which you will learn more about in a later chapter. For now, simply note that an API endpoint is a website address. This address isn't meant for you to visit as a human, but is meant for other computer programs to visit. The following Alpha Vantage page provides documentation for all of the different API endpoints within its system: <https://www.alphavantage.co/documentation/>.

We will explore the TIME\_SERIES\_DAILY endpoint, which grants access to the open, close, high, and low prices for each day a stock traded, as well as information about the total volume of trades that occurred each day for the stock. For example, the following API endpoint would provide all daily price and volume data for Google (i.e., ticker symbol GOOG). If you click the link, you can see what your computer will "see".

[https://www.alphavantage.co/query?function=TIME\\_SERIES\\_DAILY&datatype=json&symbol=GOOG&outputsize=full&apikey=demo](https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&datatype=json&symbol=GOOG&outputsize=full&apikey=demo)

If you clicked the link, you will note that the output isn't meant for human consumption. The data you see is in a format called JSON (JavaScript Object Notation), which is read by computer programs. You can also request the data to be returned in CSV format, which you could save to your computer using Python libraries.

To convert either format to a pandas data frame to take advantage of the many tools available through the pandas library, you will have to write some code. Below, you are provided with a simple class that will request data for a particular ticker and then convert the data to either a Python dictionary or a pandas data frame. You could add more to this class to save a CSV file or convert a saved CSV file to a pandas data frame. Don't worry about the details of the code below at this point. You will learn more about the requests library and json library present in the code below in later chapters. For now, feel free to copy and paste the code to help you access stock data through the Alpha Vantage API.

```
import requests
import json

#simple class to get stock data from AlphaVantage API
class AlphaVantageConnector:
    def __init__(self, APIKey, dataType="json"):
        self.APIKey = APIKey
        self.dataType = dataType
        self.data = None

    def getDaily(self, ticker, outputSize="full"):
        self.frequency = "daily"

        if self.dataType != "json":
            self.dataType = "csv"

        if outputSize != "full":
            outputSize = "compact"

        response = requests.get(
            f"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={ticker}&datatype={self.dataType}&outputsize={outputSize}&apikey={self.APIKey}")

        if response.ok:
            self.data = response.content
            return self.data
        else:
            raise Exception(f"{response.status_code}: {response.reason}")

    def toDictionary(self):
        if self.data is None:
            raise Exception("Cannot create dictionary. No data was retrieved from the API.")

        if self.dataType == "json":
            if self.frequency == "daily":
                dictionary = json.loads(self.data)
                return dictionary['Time Series (Daily)']
```

```

def toDataFrame(self):
    if self.data is None:
        raise Exception("Cannot create data frame. No data was retrieved from the API.")

    if self.dataType == "json":
        df = pd.DataFrame.from_dict(self.toDictionary()).transpose().rename(
            columns={'1. open': 'Open', '2. high': 'High', '3. low': 'Low', '4. close': 'Close', '5. volume':
            'Volume'})
        df.index.name = "Date"
    return df

#how to use the AlphaVantageAPI class to collect stock data
av = AlphaVantageConnector("AlphaVantageAPIKeyGoesHere")
av.getDaily("META") #META is the stock ticker for the company Meta
data = av.toDataFrame() #convert the AlphaVantage data to a pandas data frame
data.head() #use the data frame in different ways as outlined in this chapter

```

At the bottom of this code example below, you will see the part of the code you will actually need to change based on the particular ticker symbol you want to examine. Now you can begin working with actual stock data.

## Descriptive Statistics and Visualizations in Python

Once you read in data sets, manage the data types, and handle strange data, outliers, and missing values, you can begin to visualize your data and run statistical methods to identify trends in your data. This text is meant as an introduction to programming for data analytics and does not explore statistical methods in depth. This section will introduce some very simple statistics that you can use, as well as some visualizations you can plot to examine your data.

### *Descriptive statistics*

This text doesn't focus heavily on predictive statistics and other advanced statistical methods. Such topics are covered in depth in texts about data mining and machine learning. This chapter focuses on simple descriptive statistics. Descriptive statistics explain what your data looks like. You have already been introduced to several types of descriptive statistics in this chapter. For example, **measures of central tendency**, such as the mean and median, explain the data by providing a single value to represent the "center" of the data points. **Measures of dispersion**, such as quartiles and standard deviations, were also introduced to explain how data values are spread. Basic descriptive statistics can be printed out for each column in your data set by using the `describe()` method in pandas. For example:

`data.describe()`

For a data set with Age and Salary columns (i.e., features), the results might look like Figure 6.4.

As a reminder, you can also call individual pandas methods to get similar information. For example, the `mean()` method will give you the mean value, the `median()` method will give you the median value, and

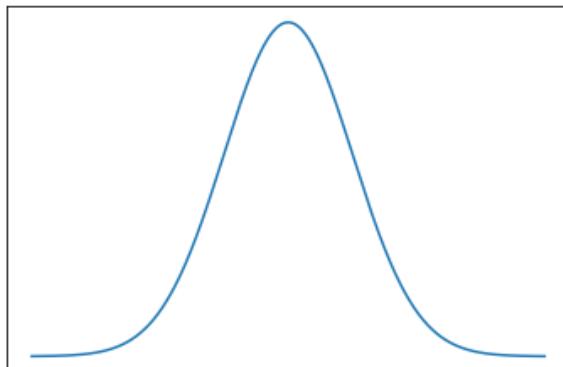
the **mode()** method will give you the mode value (i.e., the most frequent value). These methods provide different measures of central tendency. The **min()** and **max()** methods will give you the min and max values of a column, the **std()** method will give you the standard deviation (i.e., a standardized measure of dispersion), and the **var()** method will give you the variance of the data (i.e., an unstandardized measure of dispersion based on raw scores). These methods provide different measures of dispersion. The **count()** method will tell you how many non-null values are in a column/data frame. Together, these statistics can give you a sense of what your data looks like.

**Figure 6.4: Example of Descriptive Statistics**

	Age	Salary
<b>count</b>	48.00	45.00
<b>mean</b>	50.15	85263.44
<b>std</b>	23.99	34951.95
<b>min</b>	16.00	29841.00
<b>25%</b>	26.25	58846.00
<b>50%</b>	53.50	82509.00
<b>75%</b>	68.00	110981.00
<b>max</b>	93.00	195000.00

If you've taken a statistics course, you may have heard of the terms: **bell curve**, **normal distribution**, or **Gaussian normal distribution**. These terms refer to the way that many types of data tend to be distributed. Many commonly used statistical methods assume that data sets conform to the normal/Gaussian distribution. Figure 6.5 shows a Gaussian distribution. Two methods that you might be interested in that relate to the shape of the data distribution are the **skew()** method and the **kurtosis()** method.

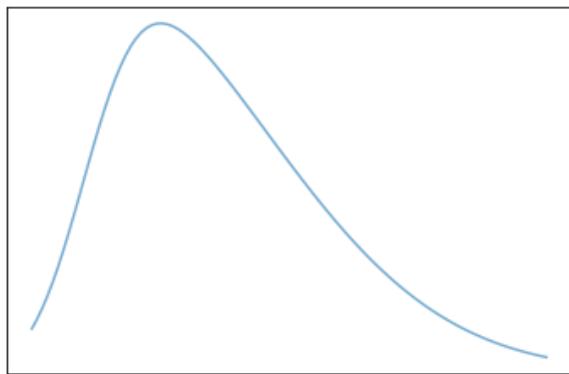
**Figure 6.5: Gaussian/Normal Distribution**



The **skew()** method measures the skewness of a distribution. **Skewness** is a measure of the asymmetry of a distribution. Figure 6.6 shows a distribution that is positively skewed. Skewness can be caused by

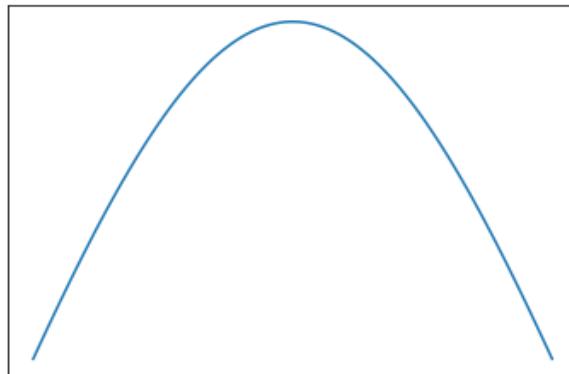
outliers on one side of a distribution or more higher than lower values or vice versa. Extreme skewness can create issues for some statistical methods.

**Figure 6.6: Skewed Distribution**



The kurtosis() method measures the kurtosis of the distribution. **Kurtosis** measures extreme values at the tails of the distribution of your data that can make the distribution look flatter or more peaked. Kurtosis is sometimes called tailedness because it is concerned with how values are distributed at the two tails of the distribution. Figure 6.7 shows a distribution with kurtosis. Again, extreme kurtosis can cause incorrect analyses for some statistical methods.

**Figure 6.7: Distribution with Kurtosis**



Although many statistical methods are robust to relatively small amounts of skewness and kurtosis, some methods are sensitive to larger deviations in the normality of your data's distribution. Read about the statistical methods you use to identify how sensitive they are to skewness, kurtosis, and other aspects of your data distribution. You can use the skewness and kurtosis values from the skew() and kurtosis() methods to compare against the acceptable values of skewness and kurtosis for a particular statistical method.

## **Exploring Basic Relationships in Data**

In addition to understanding the distribution of data points in each column (i.e., for each feature). You might also be interested in **how data in different data columns are related to one another**. Although there are many complex statistical methods that can be used to examine relationships between different columns/features within data, such as regression analysis, cluster analysis, and support vector machines,

you can get started with simpler statistical methods and visualizations. This text focuses on the basics of exploring relationships between data columns using Python, such as through ***correlations*** and ***covariances***. Basic plots can also be created to visualize simple relationships. However, these simple methods alone should not be used to make real-world decisions. If you want to understand business data and be able to make predictions to inform decision making, you need to learn more about statistics.

## **Scatter plots**

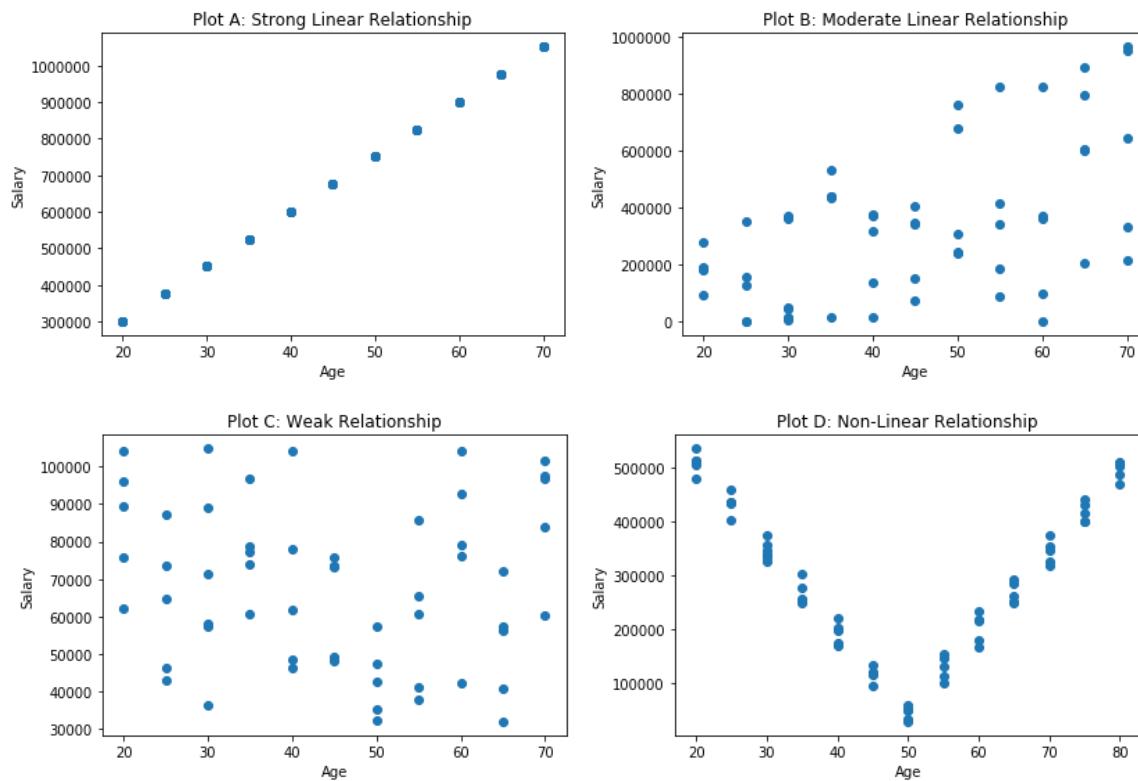
To begin, let's learn how to visualize the relationships between two data columns. Scatter plots can offer a useful first visual indicator of the existence of relationships between data columns. **Scatter plots** show each data point in the dataset with an (x, y) coordinate on a cartesian plane where x represents one column and y represents another column. In many scientific disciplines, x is called the ***independent variable*** and y is called the ***dependent variable***. Many scientific studies assume that changes in x cause changes in y. However, not all statistical approaches make this assumption of causality. x and y could just be variables that are related to one another, possibly with some other variable z that causes the relationship between x and y.

For example, assume you have a data set with ages and salaries. Assume that you are a human resource employee and have been asked how strong the relationship is between an employee's age and the employee's salary. Salary tends to increase with age; however, that does not mean that age causes salary to increase. Rather, as you age, you gain more experience which is valued and often equates to higher salaries (assuming the experience you have is valued within the economy). The scatter plots in Figure 6.8 show different possibilities for the relationship between age and salary, though some are certainly not likely to occur in the real world.

Plot A in Figure 6.8 shows a ***strong linear relationship*** between age and salary, such that as age increases as salary increases. The opposite could have been graphed as well with a downward sloping line showing that salary decreases with age. A ***moderate linear relationship*** is depicted in Plot B. Although there seems to be a positive linear trend between age and salary in Plot B, it isn't as clear and strong as the relationship in Plot A. Plot C shows a ***weak relationship***. The data seems to be randomly spread out, such that no discernable relationship exists between age and salary. Plot D shows a relationship, but the relationship is not linear (a.k.a., a ***non-linear relationship***). It seems to show a V-shaped relationship, such that employees start with a high salary, decrease in salary as they age until the age of 50 and then steadily increase in salary as they age past 50. Again, these are just hypothetical examples and not reflections of the real world.

Some relationships between features will be linear and others will be non-linear. Some statistical methods are designed to measure linear relationships, but perform poorly when analyzing non-linear relationships. For example, the data in Plot D might show no relationship in a model designed to analyze linear relationships. For this reason, it can be important to know whether your data follows a linear pattern or a non-linear pattern.

**Figure 6.8: Example Scatter Plots Showing Different Relationships**



The visualizations above were created using the ***matplotlib library*** in Python. Matplotlib is another useful analytics library within Python that can be used alongside pandas and other analytics libraries.

Assuming you had an Excel spreadsheet with the data reflected in the scatter plots in Figure 6.8, the following code would allow you to read the data into a notebook and generate scatter plots. The code relies on pandas and pyplot (a module within the matplotlib library). The code below assumes that the spreadsheet contains a column called Age and a column called Income. The plots in Figure 6.8 were created by changing the values in the Income column. For more information about the scatter(x,y) method that is used to generate scatter plots, visit:

[https://matplotlib.org/3.1.3/api/\\_as\\_gen/matplotlib.pyplot.scatter.html](https://matplotlib.org/3.1.3/api/_as_gen/matplotlib.pyplot.scatter.html).

```
import pandas as pd
import matplotlib.pyplot as plt
data = pd.read_excel("C:/path/to/file/dataset.xlsx")

plt.scatter(data.Age, data.Income) #creates a scatter plot based on column x and y
plt.xlabel("Age") #creates a label for the x-axis that is displayed below the plot
plt.ylabel("Salary") #creates a label for the y-axis that is displayed to the left of the plot
plt.title("Relationship Between Age and Salary") #creates a label for plot at the top of the plot
plt.show() #shows the plot as output of the notebook cell
```

## Plotting time series data

Scatter plots show a relationship between two columns. However, many business analytics applications call for the examination of changes in a single column (i.e., stock price data) over time. Data that you examine across multiple time points is called **time series data**. Time series analysis seeks to explain or predict changes in data over time. For example, analyzing and predicting changes in the stock price for a particular company. Again, this text doesn't focus on the complexities of time series analysis. Like regression analysis, entire texts and courses are dedicated to teaching time series analysis. This section is meant to provide a foundation of how to visualize time series data in Python. However, Python does provide many libraries to make predictions from time series data.

```
import pandas as pd  
import matplotlib.pyplot as plt
```

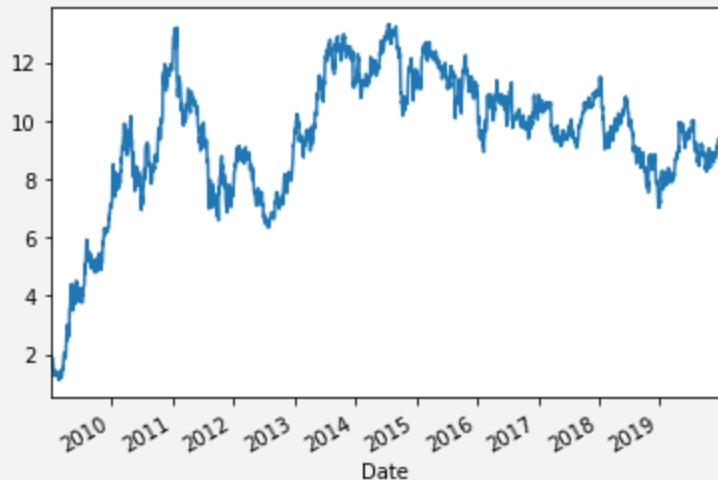
Let's assume you wanted to look up daily stock price data for Ford (ticker symbol F). To get the data, you would need to copy the AlphaVantageConnector class referenced earlier in this chapter to use it to collect stock data. The code below assumes that the AlphaVantageAPI class has been copied.

```
#the following line reads data from Alpha Vantage API to get Ford (i.e.,F) price and volume data  
av = AlphaVantageAPI("AlphaVantageAPIKeyGoesHere")  
av.getDaily("F")  
fData = av.toDataFrame()
```

The data frame created by AlphaVantageConnector() will include several columns: Date, High, Low, Open, Close, and Volume. We will plot the Close column. To plot the daily closing price, you would write the following code:

```
fData["Close"].plot() #plot the adjusted closing price of the stock
```

The plot() method would produce the following price plot:



If you have a finance background and know how to manipulate stock data, you can now extract stock data easily from a data provider like Yahoo Finance through pandas-datareader or Alpha Vantage to perform calculations to build strong portfolios, predict stock prices, etc. Knowing how to write analytics applications allows you to differentiate yourself by analyzing data with new models and algorithms.

## Correlation

To this point, you have learned how to visualize relationships between two columns or one column over time. Additionally, simple descriptive statistics can be calculated to identify relationships between columns. As mentioned before, these simple statistics do not necessarily denote the existence of causality between the columns (i.e., changes in column x cause changes in column y). Rather, these statistics denote the existence of a relationship that may or may not be causal. Experiments and repeated studies are often needed to determine causality, and even these experiments and studies can contain flawed assumptions about causality.

Correlation is a simple descriptive statistic that provides insight to the strength of relationships between two columns. Although the term correlation refers to a variety of relationships between variables, the term is often associated with the Pearson correlation coefficient. The **Pearson correlation coefficient** measures linear relationships between two columns. Correlation measures whether changes in one variable are related to changes in another variable. **Correlation coefficient values lie between -1 and 1.** If two variables have a perfect and positive linear relationship (i.e., data points of their (x, y) coordinates fall on a straight line sloping upwards), the two columns will have a correlation of 1. If they have a perfect and negative linear relationship (i.e., data points of their (x, y) coordinates fall on a straight line sloping downwards), the two columns will have a correlation of -1. If the (x, y) coordinates are scattered at random, the two columns will have a correlation near 0. Moderate relationships will have values of 0.50 or -0.50 depending on whether the linear relationship is positive or negative.

If you have multiple columns and you want to examine the correlations between all of the different column to column combinations, you can create a correlation matrix. For example, assume you had three columns named: Column A, Column B, and Column C. You could create a correlation matrix showing the relationship between: Column A and B, Column A and C, and Column B and C. The matrix would look something like the matrix depicted in Figure 6.9. The diagonals are 1 because the correlation between Column A and Column A, Column B and Column B, and Column C and Column C is a perfect correlation. That is, each column is perfectly correlated with itself. Only one half of a correlation matrix is needed. The other half is simply a mirror image across the diagonal. You would only need to examine the green values in Figure 6.9 to determine the correlations between each of the columns. The red values simply provide the same values again.

**Figure 6.9: A Correlation Matrix**

	Column A	Column B	Column C
Column A	1.0	0.7	0.2
Column B	0.7	1.0	-0.3
Column C	0.2	-0.3	1.0

Correlation coefficients can be calculated by using the corr() method in pandas.

Assume you had a dataset with the Age, Salary, and Household Income of your employees. If you wanted to know if there was a relationship between any of these data columns, you could use the corr() method to calculate the correlations.

```
import pandas as pd  
  
data = pd.read_excel("C:/path/to/data/Data.xlsx")
```

To create a correlation matrix with the data, you could simply type:

```
data.corr()
```

Based on made up data used for this example, the resulting correlation matrix looked like this:

	Age	Salary	Household Income
Age	1.00	0.07	0.02
Salary	0.07	1.00	0.94
Household Income	0.02	0.94	1.00

The data suggests a weak linear relationship between Age and Salary (0.07), a weak relationship between Age and Household Income (0.02), but a strong linear relationship between Salary and Household Income (0.94). As in this example, strong correlations aren't always informative. It is fairly obvious that a high salary will have a strong relationship with household income. In fact, it could be said that the two features might be redundant in this case.

You can also calculate the correlation for just two columns without producing an entire correlation matrix. To examine the correlation between Age and Household Income, you could do the following:

```
data.Age.corr(data["Household Income"])
```

This code would output the same 0.02 value as depicted in the correlation matrix above, except without rounding the correlation value (i.e., 0.02476512755017399).

## Covariance

Another statistic that measures the strength of relationship between two columns is covariance. Correlation is actually derived from covariance calculations. Correlation is calculated by dividing the covariance of the two data columns by the standard deviations of each column multiplied together. Correlations are often preferred by those new to statistics because they are more easily interpreted (i.e., values between -1 and 1 with 1 and -1 being perfect relationships and 0 being no relationship). Correlation is a standardized measure of covariance. Covariances are not as easy to interpret, but are used in a number of more complex statistical methods, such as structural equation modeling and some machine learning models.

In a correlation matrix, the diagonals of the matrix were equal to 1 because the correlation of a column with itself is a perfect linear relationship. However, in a covariance matrix, the diagonals represent the variance of each column.

Assume that you want to calculate covariances for the same data set examined in the correlation section. This could be accomplished with the cov() method from pandas..

```
import pandas as pd  
  
data = pd.read_excel("C:/path/to/data/Data.xlsx")
```

To create a covariance matrix with the data, you could simply type:

```
data.cov()
```

Based on made up data used for this example, the resulting covariance matrix looked like this:

	Age	Salary	Household Income
Age	430.17	52521.24	29356.00
Salary	52521.24	1211792706.89	1872186296.33
Household Income	29356.00	1872186296.33	3266435349.96

Interpreting the above data is difficult.

As stated above, the diagonals are the variance for each column. Like the correlation matrix, only one half of the matrix is needed. The other half provides redundant information.

You can also calculate the covariance for just two columns without producing a covariance matrix. To examine the covariance between Age and Household Income, you could do the following:

```
data.Age.cov(data["Household Income"])
```

This code would output the same 29356.00 value as depicted in the covariance matrix above, except without rounding the covariance value (i.e., 29355.99970744681).

Although covariance is difficult to interpret, it is used in a number of formulas. We will examine some such formulas in later chapters. You can build upon the simple knowledge provided in this chapter to write complex statistical algorithms to help you make better business decisions. Again, it is in your best interest to take some additional statistics courses, such as regression, econometrics, probability, or others.

## Chapter 6 Assignment

Please see the [Chapter Assignments Workbook](#) for the Chapter assignment.

# Chapter 7:

## Integrating Databases into a Program

To this point, we have largely seen and created simple programs that store information within data structures, such as arrays and maps, in computer memory (i.e., random access memory: RAM). Information stored in RAM can be retrieved very quickly as compared to information stored on certain types of hard drives. The problem with random access memory is that it is volatile. In terms of memory, the term **volatile** means that when a computer loses power, the information in memory is lost. There also tends to be less storage space in RAM than on a hard drive. For these reasons, most databases store information on hard drive technology.

Some emerging database technologies, called “in-memory” databases, offer an exception to this rule. In-memory databases store data in computer memory. But, most traditional, industry standard databases store data on hard drives. Hard drives ensure that data is persisted even when the computer loses power. Hard drives also offer ample storage space. Persistent data storage is essential if you plan to develop business systems that track customers, shipments, employees, or other business objects over extended time periods.

This chapter will explore the integration of database queries within computer programs. Databases store important business data. Data analytics algorithms rely on this data. Thus, understanding how databases and programs communicate is crucial to developing strong business analytics skills.

### What is a Database?

Excepting emerging database technologies, like in-memory databases, databases store data in files on hard drives, usually on servers. These data files are accessed and interpreted by a database management system. A **database management system** provides tools to create databases, manipulate data within databases, and control who can access data.

Many types of databases exist, such as columnar databases, key-value databases, document-oriented databases, and relational databases. This chapter focuses primarily on relational databases, as they continue to be the most widely used database technology in practice. However, other types of databases are increasingly common.

**Relational databases** store information in data tables. **Tables** store information about specific business objects (i.e., customers, employees, suppliers, etc.) in rows and columns, somewhat like data storage in spreadsheets. Each table represents a particular business object. For example, you might have a Customer table, a Product table, an Order table, etc. These tables are linked together with relationships. The columns of a database table, sometimes called **fields**, represent attributes of the business objects contained within a table. For example, the columns/fields of a Customer table might be: firstName, lastName, phoneNumber, address, etc. The rows of a database represent particular instances or **records** of the business object within the table. Each row is represented by a unique value called a **primary key** to ensure that each business object in the table can be accessed independently of other objects. Figure 7.1 shows an example of a database table. The customerId field acts as the primary key.

**Figure 7.1: Example of a Database Table with Fields and Primary Key**

Customer Table				
customerId	firstName	lastName	phoneNumber	address
1	Jamie	Doe	555-555-5551	123 Fake St.
2	Ryan	Doe	555-555-5552	123 Fake St.
3	Taylor	Doe	555-555-5553	8472 Doe Ln.

Data in relational database tables are accessed and manipulated through the structured query language (SQL). **SQL**, pronounced See-Quel or S.Q.L., provides methods to create databases and tables, insert data into tables, select data from tables, update data in tables, and delete data from tables amongst other functions. A variety of SQL database management systems exist, such as Oracle SQL, Microsoft SQL, and MySQL. In this text, we will make use of MySQL databases. Oracle offers a free community version of MySQL for business and educational use. MySQL also has a paid enterprise-level database management system as well with advanced data manipulation features. However, we will rely on the community version.

There are many rules about how database tables relate to one another, much like the rules for connecting classes together. This text does not go into these details. To learn more about database relations, such as normalization and denormalization, please read a database textbook or read technical blogs about SQL and database design. This text will also not describe how to write SQL code in detail. Although SQL code is much simpler than the logic required to create a computer program, the focus of this text is computer programming for business applications. The text provides examples of SQL code.

## Domain Classes and Database Tables

Although it is possible to simply connect to a database without the use of classes in Python, business systems often contain more complex architectures than this. **Software architecture** refers to the way in which you organize classes and other system components to create system functionality. Frameworks, such as the Model-View-Controller (MVC), Model-View-Presenter (MVP), and Model-View-ViewModel (MVVM) can provide an architecture for systems. These frameworks break the logic of a program into three distinct types:

- **Views:** classes containing user interface logic. You will learn more about user interface design in later chapters.
- **Controllers/Presenters:** classes containing business and calculations logic; classes that manipulate Model classes; they may also act as bridges between Views and Models
- **Models:** classes containing data about business objects; may make connections with databases

In these architectures, model classes are also called **domain classes**. Although the name “model class” and “domain class” may sound novel to you, you have actually seen and created many domain/model

classes in your chapter assignments. Any of the classes you created that represented business objects (e.g., Client, Product, Stock, etc.) are model/domain classes. The main purpose of a domain class is to store information about a business object (e.g., a customer's name, phone number, and address) in memory while the program is running. When you bring a database into the picture, a domain class is designed to store information retrieved from a database in RAM so a computer program can use the information. Domain classes are also used to temporarily store data in RAM before sending the data to a database for permanent storage. Model/domain classes can also help to ensure that data is consistent and follows certain guidelines, such as a phone number being formatted appropriately with the appropriate number of digits (e.g., 555-555-5555).

Given that domain classes are used to interact with the business objects stored in database tables, some architectures require that you create a domain class for each table in your database. For example, if you have a Customer table in the database, you would create a Customer class in your computer program; if you had a Product table in the database, you would create a Product class in your program; and so on. Additionally, the domain classes you create in your program will likely have the same attributes as their corresponding database tables. For example, if you had a Customer database table with firstName and lastName columns, you would likely create a Customer class with firstName and lastName attributes (though the exact naming of attributes doesn't need to match).

In most of the programs you have written so far, you have instantiated domain objects (e.g., Customer objects) and manually set the values of the objects' attributes. For example, you have seen a simple program with a Customer object where the firstName was set to "Jeff" and the lastName set to "Wall." In a real system, there is no need to manually set values in this manner. Instead, the attribute values either come from a database or from users through user interface objects like form elements. This chapter will show you how to move away from manually setting attribute values toward dynamically setting values based on information in a database.

## Connecting to Databases within Python Programs

To start utilizing databases in your programs, you must first learn to connect from a program to a database. Connecting to databases with Python consists of a few steps. First, you must ensure that the appropriate database driver is installed on the computer your code runs on. A **database driver** makes a connection between a computer program and a database. Database driver classes allow a computer program to send SQL code to a database to access and manipulate data in the database. The database driver often exists within the Model/Domain classes when using an MVC, MVP, or MVVM framework. For example, if you wanted to get data about a customer from a Customer database table, the database driver would connect to the database, send SQL code to the database to select the appropriate data from the table, and accept the results of the database query. The resulting data collected by the database driver might then be passed to a Customer object to be stored in memory and presented to the user through the user interface.

Database drivers are often provided either through the programming language's default libraries or downloaded or imported into a program. The database driver we will use to access MySQL databases is called the MySQL Python connector library.

Sometimes, developers choose to access the functionality of a database driver through a custom built class called a DatabaseGateway class. A DatabaseGateway class allows developers to change to a

different database driver if a database is replaced. We will not go deep enough into architecture to consider database gateways. Simply, recognize that system architectures are complex. This chapter will start with a simple example and then examine slightly more complex architectures.

The MySQL Python connector (i.e., the database driver for MySQL databases using the Python programming language) can be downloaded at: <https://dev.mysql.com/downloads/connector/python/>. Even simpler, you can download and install the MySQL connector by opening the Anaconda Prompt or Terminal (for Mac) instead of the Anaconda Navigator and typing the single line of code below. The Anaconda Prompt can be accessed by typing “Anaconda Prompt” in the Windows search bar next to the Start Menu. The Terminal can be used on Mac or Linux machines. Once the prompt opens, you simply type:

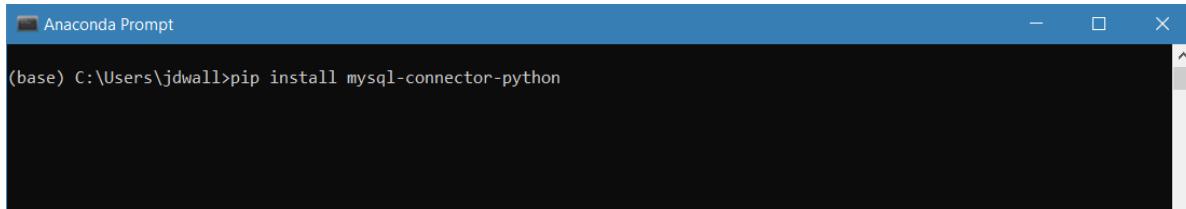
```
pip install mysql-connector-python
```

Depending on how your system is configured, you might be using pip3, which would change the command to:

```
pip3 install mysql-connector-python
```

Hit enter and the mysql connector library for Python will be installed on your machine. Figure 7.2 shows how the Anaconda Prompt can be used to install the MySQL connector library into Python. The Terminal on Mac will look similar.

**Figure 7.2: Installing MySQL Connector for Python with Anaconda Prompt**



For the purposes of this text, we have examined Python as a programming language for conducting data analytics. You were introduced to several Python libraries, such as pandas, to help you process business data. Databases are an important component in the process of analyzing data. As such, this section will show you how to access a database to manipulate data for analytics programs in notebooks.

To connect to a database via Python, you will need to know: 1) the **database host**: such as the IP address of the database server, the domain name of the server, or localhost for databases on the same server as your code; 2) the **name of the database** that exists on the host; 3) a **username** with permissions to access the database; 4) a **password** for the user.

To start, import mysql.connector into the notebook. This import gives your notebooks access to the functionality provided by the mysql connection library. Then, you can use the connect() method to connect to a database as in the code below.

### Using mysql.connector to Connect to a Database

```
import mysql.connector

class Customer:
    def getCustomerData(self):
        try:
            connection = mysql.connector.connect(
                user='myUser',
                password="myPassword",
                host="localhost",
                database='mis2100example_db' )

            connection.close()
        except mysql.connector.Error as err:
            print(err)
```

Notice in the example that the database connection is wrapped in a try-except statement. This is common for database connections, as database servers or the networks used to connect to the database server can go down. For example, the try-except statement allows you to add an alternate database to connect to in the except statement if a failure occurs with the database in the try statement.

Now that you have seen how to connect to a database, we can explore the basic database functions of inserting, selecting, updating, and deleting data. The database connection object provides the ability to perform these functions. These basic functions are sometimes called **CRUD functions** (create, read, update, and delete). To begin, you must understand that each of these basic functions is performed by a statement.

## SQL Statements and Prepared Statements

Databases execute SQL statements to retrieve, add, manipulate, or delete data within tables. These SQL statements can be fairly simple or quite complex. In a program, a SQL statement is represented by a string. For example, the following SQL statement stored in a variable named “query” would select the firstName and lastName fields for each row from a table named Customer:

```
query = "SELECT firstName, lastName FROM Customer"
```

Simple statements don’t require any variables to be embedded into the statement. Complex statements require variables to be added to the SQL statements. A simple SQL statement might ask the database management system to return all customers from the Customer table, such as in the example above. A more complex statement might ask the database management system to return all customers from the Customer table whose last name is equal to the name provided in a variable.

When embedding SQL statements into a program, you need to be cautious about how you write SQL statements. Simple statements don’t require any special treatment. However, when you need to embed a variable into a SQL statement, you should use a prepared statement instead of simply embedding the

variable into the SQL statement string. It is possible to simply embed variables into the SQL statement string. Unfortunately, this can create security vulnerabilities in a program (i.e., SQL injection attack vulnerabilities). Instead of embedding variables directly into a SQL statement string, the MySQL Python connector library provides a way to embed variables securely.

Let's start by looking at a simple SQL statement that doesn't require any variables in the SQL statement. First, you connect to a database. In Python, the object that manages SQL statements is called a **Cursor object**. Second, you create a new Cursor object using the cursor() method of the connection object. Third, you create a SQL statement string that will be executed by the Cursor object. Last, you call the execute() method on the cursor object and pass the statement string as an argument. Once you have used the results of the Cursor execution, you can close the Cursor object to clear the results and then close the database connection. The example below shows this process.

#### Example of Executing a SQL Statement

```
import mysql.connector

class Customer:
    def getCustomerDatabaseConnection(self):
        try:
            connection = mysql.connector.connect(
                user='myUser',
                password="myPassword",
                host="localhost",
                database='mis2100example_db')

            return connection
        except mysql.connector.Error as err:
            print(err)

    def getCustomers(self):
        #get a MySQL connection object from the getCustomerDatabaseConnection() method
        connection = self.getCustomerDatabaseConnection()

        #create a cursor object by calling the cursor() method on the connection object
        cursor = connection.cursor()

        #create a statement string
        statement = "SELECT firstName, lastName FROM Customer"

        #execute the statement using the cursor object using execute() method of the cursor object
        result = cursor.execute(statement)

        #use of the database results would go here

        #close the cursor
        cursor.close()
```

```
#close the connection
connection.close()

#instantiate the Customer class and call the getCustomers() method
customer = Customer()
customer.getCustomers()
```

Although the simple statement created by the Cursor object can be used at times, prepared statements created by the Cursor object will be more useful in most use cases. More often than not, your SQL statements will include input from a user. For example, a user might want to only select data where the Customer has a firstName of Jeff and a lastName of Wall. In this case, the statement would look like this:

```
SELECT firstName, lastName FROM Customer WHERE firstName="Jeff" AND lastName="Wall"
```

In the statement above, the values “Jeff” and “Wall” would be provided by a user entering those values into a form element or as arguments in a method call, which would be embedded in a statement string. User input should **NEVER** be trusted. It is possible for a user to send data that can exploit a SQL statement to perform undesirable actions on the database (i.e., SQL injection attacks). Anytime you accept user input that will be embedded into a SQL statement, you want to use a prepared statement object. A prepared statement object has built-in protections to ensure that users cannot hijack a statement and perform SQL injection attacks. You could easily create a SQL injection attack by using f-strings like in the example below:

DO **NOT** DO IT THIS WAY!!! It can lead to injection attacks.

```
firstName = "Jeff"
lastName = "Wall"
query = f"SELECT firstName, lastName FROM Customer WHERE firstName={firstName} AND
lastName={lastName}"
```

Instead, use prepared statements. Cursor objects that support prepared statements require SQL statement strings that include placeholders everywhere user input needs to be added to the SQL statement. There are two placeholders that are accepted by the MySQL connector library: %s and ?. The statement above would look like this if it were created for a Cursor object with a prepared statement:

```
query = "SELECT firstName, lastName FROM Customer WHERE firstName=? AND lastName=?"
```

Notice that in place of the values “Jeff” and “Wall,” question marks are used in prepared statements. To create a prepared statement, you must pass the following argument to the Cursor object when it is created: `prepared=True`. Once you create a Cursor for a prepared statement, you can replace the placeholders in the prepared statement string within actual data by passing an extra argument to the `execute()` method of the Cursor object. Data values are added in the order that the question marks appear in the statement. The code below shows how a prepared statement is created and executed.

### Example of Executing a Prepared Statement

```
import mysql.connector

class Customer:
    def getCustomerDatabaseConnection(self):
        try:
            connection = mysql.connector.connect(
                user='myUser',
                password="myPassword",
                host="localhost",
                database='mis2100example_db' )

            return connection
        except mysql.connector.Error as err:
            print(err)

    def getCustomerDataByName(self, firstName, lastName):
        try:
            #get a MySQL connection object from the getCustomerDatabaseConnection() method
            connection = self.getCustomerDatabaseConnection()

            #create the parameterized prepared statement string using a ? in place of actual data
            statementString = "SELECT * FROM Customer WHERE firstName=? AND lastName=?"

            #create a cursor for a prepared statement
            cursor = connection.cursor(prepared=True)

            #execute accepts data values to replace the ?'s in the statement.
            cursor.execute(statementString, (firstName, lastName))

            #close the cursor and the connection
            cursor.close()
            connection.close()
        except mysql.connector.Error as err:
            print(err)

#instantiate the Customer class and call the getCustomerDataByName() method
customer = Customer()
customer.getCustomerDataByName()
```

With a basic understanding of how statements and prepared statements work, we can now explore the basic CRUD functions that you can perform on a database table.

## **Inserting data into a database table (create)**

Inserting data into a database table occurs through SQL insert statements. Insert statements almost always accept user input that is to be stored in the database. As such, **a prepared statement should almost always be used for insert statements**. SQL provides several syntax options to insert data into a database. We will explore the “set” insert method, as it mimics the syntax for the update statement. This will simplify what you need to understand about SQL statements. The basic syntax for a SQL insert statement is as follows:

```
INSERT INTO tableName SET column1Name=column1Value, column2Name=column2Value,  
columnNName=columnNValue
```

Let's examine a more specific example that creates a new customer record in a database table named Customer. Assume that the table has the following columns: firstName, lastName, and phoneNumber. The following SQL statement would insert a new record into the Customer table.

```
INSERT INTO Customer SET firstName="Jamie", lastName="Doe",  
phoneNumber="555-555-5551"
```

For this example, the data will be passed into a method as method arguments. When inserting data into a database, you often want to know what id the database provided to the new entry. The value of the database table id can be returned by calling the lastrowid attribute on the Cursor object: `cursor.lastrowid`.

You will notice that the insert statement must be committed by the Connection object after the `execute()` method is called on the Cursor object. To commit changes to the database simply call the `commit` method: `connection.commit()`. To make sure that changes were actually made to the database, you can access the `rowcount` attribute on the Cursor object like this: `cursor.rowcount`. The `rowcount` attribute stores how many rows were affected by a committed SQL statement. If the `rowcount` is 0, then you know the statement didn't work correctly. Let's examine some code that would allow you to insert data into a database table.

### **Example of Inserting Data into a Database**

```
import mysql.connector  
  
class Customer:  
    def getCustomerDatabaseConnection(self):  
        try:  
            connection = mysql.connector.connect(  
                user='myUser',  
                password="myPassword",  
                host="localhost",  
                database='mis2100example_db'  
  
            return connection  
        except mysql.connector.Error as err:
```

```

print(err)

def saveCustomer(self, firstNameP, lastNameP, phoneP):
    try:
        #get a MySQL connection object from the getCustomerDatabaseConnection() method
        connection = self.getCustomerDatabaseConnection()

        #create the insert statement with parameters (i.e., ?) for a prepared statement
        statementString = "INSERT INTO Customer SET firstName=?, lastName=?,  

                           phoneNumber=?"

        #create a cursor for a prepared statement
        cursor = connection.cursor(prepared=True)

        #execute accepts data values to replace the ?'s in the statement.
        cursor.execute(statementString, (firstNameP, lastNameP, phoneP))

        #commit the changes to the database
        connection.commit()

        #if the execution worked, set the attributes of the class with the data sent to the database
        if cursor.rowcount > 0:
            #get the id the database created for the new row and store it in an attribute
            self.customerId = cursor.lastrowid
            self.firstName = firstNameP
            self.lastName = lastNameP
            self.phone = phoneP

        #close the cursor and the connection
        cursor.close()
        connection.close()
    except mysql.connector.Error as err:
        print(err)

#instantiate the Customer class and call the saveCustomer() method
customer = Customer()
#the values Jeff, Wall, and 555-555-5551 would be inserted into the database table
customer.saveCustomer("Jeff", "Wall", "555-555-5551")

```

Be sure to use prepared statements for insert statements. You now possess enough information to programmatically add data into a database.

## ***Selecting data from a database (read)***

Selecting information from a database table is accomplished with a SQL select statement, also called a query or query statement. The basic syntax for a select statement is:

**SELECT** list of columns to return **FROM** table name **WHERE** conditions to filter results

You saw an example of select statements in the earlier section about statements and prepared statements. For reference, here is an example of a complete select statement:

```
SELECT firstName, lastName FROM Customer WHERE firstName="Jeff" AND lastName="Wall"
```

If you wish to select all of the fields from a table, you can use the \* symbol instead of providing a list of fields. For example:

```
SELECT * FROM Customer
```

As in the previous example, you must start with a connection, create a statement string, create a Cursor object from the connection and execute the statement. When executing a select statement, the database will return all of the database results that fit the query parameters. You will often want to loop over these results to print to the screen or utilize them in some other fashion. The fetchall() method of the Cursor object will pull all of the database results for you to loop over as in the example below. If you are searching by a unique id value, such as a customer id, you will only expect one result. Singular results can be obtained by calling the fetchone() method instead of the fetchall() method.

The fetchone() and fetchall() methods return tuple data structure, for example:

```
(id, firstName, lastName, phoneNumber) = cursor.fetchall()
```

A **tuple** is like a list data structure, but the tuple cannot be updated or changed. The values are immutable. The fetchall() and fetchone() methods will fill the tuple with as many values as there are fields in the SELECT query. So if you selected two fields (i.e., SELECT firstName, lastName...), the tuple would possess two values in that order: (firstName, lastName). The example below selects four fields and stores those four fields in the four spaces in the tuple.

#### Example of Selecting Data from a Database

```
import mysql.connector

class Customer:
    def getCustomerDatabaseConnection(self):
        try:
            connection = mysql.connector.connect(
                user='myUser',
                password="myPassword",
                host="localhost",
                database='mis2100example_db')

            return connection
        except mysql.connector.Error as err:
            print(err)
```

```

def getCustomerDataByLastName(self, lastName):
    try:
        #get a MySQL connection object from the getCustomerDatabaseConnection() method
        connection = self.getCustomerDatabaseConnection()

        #create the parameterized prepared statement string using a ? in place of actual data
        statementString = "SELECT id, firstName, lastName, phoneNumber FROM Customer WHERE
                           lastName=?"

        #create a cursor for a prepared statement
        cursor = connection.cursor(prepared=True)

        #execute accepts data values to replace the ?'s in the statement.
        #when only one parameter (i.e., ?) exists in the statement, a floating comma is needed.
        cursor.execute(statementString, (lastName,))

        print("Your search returned the following results:")

        #loop over the results stored in the Cursor object.
        #the results are stored in variables named after the database fields
        for (id, firstName, lastName, phoneNumber) in cursor.fetchall():
            print(f"{firstName} {lastName} with id {id} can be reached at {phone}")

        #close the cursor and the connection
        cursor.close()
        connection.close()
    except mysql.connector.Error as err:
        print(err)

#instantiate the Customer class and call the getCustomerDataByLastName() method
customer = Customer()
customer.getCustomerDataByLastName()

```

From the examples above, you now have enough knowledge to retrieve information from a database. If you are using a database in your organization for data analytics purposes, you could retrieve the results of the query and process the data to answer business questions.

### ***Reading SQL data into a pandas data frame***

The pandas data library in Python has tools to help you convert data from a SQL query into a pandas data frame. By converting the SQL data to a data frame, you can use pandas to clean the data for later analysis. In Chapter 6, you learned how to import data into a pandas data frame from spreadsheets. Importing data from a SQL database can be done in a similar fashion with a single pandas method: `pandas.read_sql()`. The code example below shows how to use the `read_sql()` method to convert SQL query results into a pandas data frame.

## Convert SQL Results into a pandas Data Frame

```
import mysql.connector
import pandas as pd

try:
    #connect to the database as in the previous example
    connection = mysql.connector.connect(
        user='myUser',
        password="myPassword",
        host="localhost",
        database='mis2100example_db')

    #create the query string to retrieve that desired data
    query = "SELECT * FROM Customer"

    #create a pandas data frame in a variable named df
    #read in data from the query using the database connection created earlier
    #set the index of the dataframe to the primary key of the database table
    df = pd.read_sql(query, con=connection, index_col="customerId")

    #the pandas data frame can now be used to clean the data from the database

    #close the connection when you are finished
    connection.close()
except mysql.connector.Error as err:
    print(err)
```

Notice in the example above that the `read_sql()` method requires a SELECT statement and a database connection. By default, pandas data frames create an index column in the data to create unique ids for each row in the data frame. Often, you will want this index to be set to the primary key of the database table you are selecting from. To convert the pandas data frame index to the primary key from the database table, the `index_col` attribute must be set in the `read_sql()` method. The `index_col` should be set equal to the name of the primary key of the database table you are selecting from. In the example above, the primary key of the Customer table is assumed to be `customerId`, so the `index_col` is set to `customerId`. Once converting the SQL data to a pandas data frame, you can view and clean the data for further analysis.

## ***Updating data in a database (update)***

Updating information in a database is similar to inserting data into a database. The update SQL statement looks similar to the insert statement provided earlier. Unlike the insert statement, an update statement should almost always include a WHERE clause. The WHERE clause identifies which specific entry should be updated. If you forget to set a WHERE clause, your update statement could change values for every record in the database, which is rarely what you want. So, remember to add a WHERE clause to update queries. Like the insert statement, update statements often rely on data from a user, which means you

should use a prepared statement instead of a simple statement. The basic syntax for an update statement is as follows:

```
UPDATE table name SET column1Name = column1Value, column2Name = column2Value,  
columnNName = columnNValue WHERE conditions for which entry to update
```

Building on the Customer table example in previous sections, the query below updates an existing Customer whose id (i.e, the primary key) is equal to 1. Notice that the update statement doesn't require values for every field in the database table. In this case, we are assuming that only the last name and phone number of the customer need to be changed. The firstName is not updated.

```
UPDATE Customer SET lastName="Doe-Smith", phoneNumber="555-555-5552" WHERE id=1
```

Let's explore the code below that could be used to update a Customer entry. You will notice that the update statement must be committed even by the Connection object even after the execute() method is called on the Cursor object. To commit changes to the database simply call the commit method: `connection.commit()`. To make sure that changes were actually made to the database, you can access the rowCount attribute on the Cursor object like this: `cursor.rowCount`. The rowCount attribute stores how many rows were affected by a committed SQL statement. If the rowCount is 0, then you know the statement didn't work correctly.

#### Example of Updating Data in a Database

```
import mysql.connector

class Customer:
    def __init__(self, id , firstNameP, lastNameP, phoneP):
        self.customerId = id
        self.firstName = firstNameP
        self.lastName = lastNameP
        self.phone = phoneP

    def getCustomerDatabaseConnection(self):
        try:
            connection = mysql.connector.connect(
                user='myUser',
                password="myPassword",
                host="localhost",
                database='mis2100example_db' )

            return connection
        except mysql.connector.Error as err:
            print(err)

    def updateCustomer(self, firstNameP=None, lastNameP=None, phoneP=None):
        #fill the method parameters with data from the object if values aren't provided
```

```

#this will ensure the statement has data values, while changing the data provided
if firstNameP is None:
    firstNameP = self.firstName

if lastNameP is None:
    lastNameP = self.lastName

if phoneP is None:
    phoneP = self.phone

#get a MySQL connection object from the getCustomerDatabaseConnection() method
connection = self.getCustomerDatabaseConnection()

#create the update statement with parameters (i.e., ?) for a prepared statement
statementString = "UPDATE Customer SET firstName=?, lastName=?,
    phoneNumber=? WHERE id=?"

#create a cursor for a prepared statement
cursor = connection.cursor(prepared=True)

#execute accepts data values to replace the ?'s in the statement.
cursor.execute(statementString, (firstNameP, lastNameP, phoneP, self.customerId))

#commit the changes to the database
connection.commit()

#add the data back to attributes if arguments were provided and commit was successful
#the rowcount attribute of the Cursor tells you how many rows were affected
#a rowcount of 0 would mean that nothing was updated
if firstNameP is not None and cursor.rowcount > 0:
    self.firstName = firstNameP

if lastNameP is not None and cursor.rowcount > 0:
    self.lastName = lastNameP

if phoneP is not None and cursor.rowcount > 0:
    self.phone = phoneP

#close the cursor and the connection
cursor.close()
connection.close()
except mysql.connector.Error as err:
    print(err)

#Instantiate the Customer class with data including the id
#in a real program, the data for this object would come from a select statement result
customer = Customer("Jeff", "Wall", "555-555-5551", "45")

```

```
#only the phone is changed in this case  
customer.updateCustomer(phoneP="555-555-5553")
```

Notice in the code above that within the WindowBuilder class, a mouse click event is used to trigger the update() method. The update() method is passed the values stored inside the text fields of the user interface. With this knowledge, you can begin to add simple update features to your programs. A more complex version of this code would keep track of changes made to the text fields and only update the database fields when data values changed in the user interface.

## ***Deleting data from a database (delete)***

One last function that you may want to perform on records in a database table is the delete function. The delete function is fairly simple, but like the update statement, it should almost always be accompanied by a WHERE clause. If you fail to add a WHERE clause and other database protections are not in place, a delete statement could delete every record in a database table. Always be careful when deleting information from a database. It is best to use prepared statements instead of simple statements when writing delete statements. The basic syntax for a delete statement in SQL is:

DELETE FROM table name WHERE conditions for which entry or entries to delete

Following the Customer example in previous sections, a delete statement might look something like the query below. Notice that a WHERE clause is included that only deletes the record with an id set equal to 1. Because the id is probably the unique primary key of the table, this query would only delete one record.

DELETE FROM Customer WHERE id=23

Like the insert and update statements, delete statements must be committed with the connection commit() method. You can also call the rowcount attribute to determine if any rows were deleted. The rowcount will contain the number of rows deleted from the table. The code below shows an example of how to use the delete statement.

### **Example of Deleting Data from a Database**

```
import mysql.connector  
  
class Customer:  
    def __init__(self, id , firstNameP, lastNameP, phoneP):  
        self.customerId = id  
        self.firstName = firstNameP  
        self.lastName = lastNameP  
        self.phone = phoneP  
  
    def getCustomerDatabaseConnection(self):  
        try:  
            connection = mysql.connector.connect(  
                user='myUser',
```

```

        password="myPassword",
        host="localhost",
        database='mis2100example_db' )

    return connection
except mysql.connector.Error as err:
    print(err)

def deleteCustomer(self):
    try:
        #stop the command with an AttributeError if the customerId attribute is not set with a value
        if self.customerId is None:
            raise AttributeError("A customer id must be provided to delete a customer")

        #get a MySQL connection object from the getCustomerDatabaseConnection() method
        connection = self.getCustomerDatabaseConnection()

        #create the delete statement with parameters (i.e., ?) for a prepared statement
        statementString = "DELETE FROM Customer WHERE id=?"

        #create a cursor for a prepared statement
        cursor = connection.cursor(prepared=True)

        #execute accepts data values to replace the ?'s in the statement.
        #single parameters require a floating comma as seen below
        cursor.execute(statementString, (self.customerId,))

        #commit the changes to the database
        connection.commit()

        #if no rows were deleted, raise a mysql error, otherwise remove class attribute values
        if cursor.rowcount <= 0:
            raise mysql.connector.Error("The customer record could not be deleted from the
                                         database")
        else:
            self.customerId = None
            self.firstName = None
            self.lastName = None
            self.phone = None

        #close the cursor and the connection
        cursor.close()
        connection.close()
    except mysql.connector.Error as err:
        print(err)

```

```
#instantiate the Customer class with data including the id  
#in a real program, the data for this object would come from a select statement result  
customer = Customer("Jeff", "Wall", "555-555-5551", "45")  
customer.deleteCustomer()
```

With this example, you now have the knowledge to perform basic CRUD functions on a database via a Python notebook. Next, let's look at a more complete example.

## ***Integrating database connection objects in domain classes***

As mentioned earlier, many of the classes you have seen in your assignments and in the previous chapters were domain classes (i.e., classes meant to store data about business objects). These domain classes can be improved by integrating long-term storage within them via databases. There are different ways to integrate database functionality into a domain class.

In this chapter, you saw examples of a database driver (i.e., the MySQL connector) directly embedded into a class. Although it is easy to embed a particular database driver into a program (e.g., the MySQL connector library), this is not always the best way to design large systems. If you have hundreds or even thousands of different domain classes and each embeds the MySQL connector library directly into the class, you create a situation where transitioning to a new database would be extremely painful. Updates to hundreds or thousands of classes would be required. Software should not be so tightly coupled to databases. Databases advance, new databases arise, and costs make some databases more appealing as time goes on. If you build a system that can accept different database drivers, your system will be more adaptable over time.

One way to create flexibility in database drivers is to create a generic database driver interface. An **interface** is like a class, except that the methods of the class do not contain programming logic. Some languages like Java have distinct structures to create interfaces. Python does not have an interface structure, but does possess the ability to mimic the behavior of interfaces in a few different ways. The methods of an interface tell a developer how to design other classes that will implement the interface. Utilizing the knowledge you have, the simplest way to create the equivalent of an interface in Python is to create a class with methods that use the “pass” keyword in the body of the method. The “pass” keyword tells the Python interpreter to do nothing when the method is called.

A database driver interface might have the following methods: connect(), disconnect(), createStatement(), createPreparedStatement(), closeStatement(), commit(), etc. In the interface, the methods would not contain any logic. Rather, other classes would be created to implement the interface with specific logic. For example, you might create a MySQLDriver that inherits from the interface, then each of the methods would be overridden by the MySQLDriver to connect to a MySQL database. Later, you could create an OracleDriver that inherits from the same interface. The logic in the methods of the OracleDriver would connect to an Oracle database. Both the MySQLDriver and OracleDriver classes would use methods with the same names as outlined above.

By creating an interface, you are able to ensure that when you swap one class for another, such as swapping the MySQLDriver for an OracleDriver, the change will be minimal because the method names and parameters are the same. You can further decouple the database driver from domain classes by passing the driver into the constructor or another method of the domain class. This can also be

accomplished through more complex methods, such as dependency injection to further decouple the database driver from the domain classes. Let's see a simple example of this practice. In the code below, you will notice that an interface is created, a MySQLConnector class is created, and then a domain class is created.

### Integrating Database Drivers in a Loosely Coupled Manner

```
import mysql.connector

#creation of the database driver interface
class DatabaseDriverInterface:
    def __init__(self, hostP, databaseP, userP, passwordP):
        pass

    def connect(self):
        pass

    def disconnect(self):
        pass

    def createStatement(self, prepared=False):
        pass

    def executeStatement(self, statementString, data={}):
        pass

    def clearStatement(self):
        pass

    def commit(self):
        pass

    def raiseException(self, message):
        pass

#implementation of the interface
class MySQLConnector(DatabaseDriverInterface):
    def __init__(self, hostP, databaseP, userP, passwordP):
        self.dbConnection = None
        self.host = hostP
        self.database = databaseP
        self.user = userP
        self.password = passwordP
        self.statement = None

    def connect(self):
        try:
```

```

        self.dbConnection = mysql.connector.connect(
            user=self.user,
            password=self.password,
            host=self.host,
            database=self.database )
    except mysql.connector.Error as err:
        print(err)

    def disconnect(self):
        self.dbConnection.close()

    def createStatement(self, prepared=False):
        if not prepared:
            self.statement = self.dbConnection.cursor()
        else:
            self.statement = self.dbConnection.cursor(prepared=True)

    def executeStatement(self, statementString, data=None):
        try:
            #check if the statement is a prepared statement or not
            if isinstance(self.statement, mysql.connector.cursor_cext.CMySQLCursorPrepared):
                self.statement.execute(statementString, data)
            else:
                self.statement.execute(statementString)
        except mysql.connector.Error as err:
            print(err)

    def commit(self):
        try:
            self.dbConnection.commit()
        except mysql.connector.Error as err:
            print(err)

    def clearStatement(self):
        self.statement.close()

    def raiseException(self, message):
        raise mysql.connector.Error(message)

#domain class with database driver passed to constructor
class Customer:
    def __init__(self, dbConnectorP):
        self.customerId = None
        self.firstName = None
        self.lastName = None
        self.phone = None
        self.dbConnector = dbConnectorP

```

```

#connect to the database so the connection is available for all methods
self.dbConnector.connect()

def createCustomer(self, firstNameP, lastNameP, phoneP):
    #create the insert statement with parameters (i.e., ?) for a prepared statement
    statementString = "INSERT INTO Customer SET firstName=?, lastName=?,
        phoneNumber=?"

    #create statement (i.e., cursor), execute the statement, and commit the changes
    self.dbConnector.createStatement(prepared=True)
    self.dbConnector.executeStatement(statementString, (firstNameP, lastNameP, phoneP))
    self.dbConnector.commit()

    #if the execution worked, set the attributes of the class with the data sent to the database
    if self.dbConnector.statement.rowcount > 0:
        #get the id the database created for the new row and store it in an attribute
        self.customerId = self.dbConnector.statement.lastrowid
        self.firstName = firstNameP
        self.lastName = lastNameP
        self.phone = phoneP

        #close the cursor
        self.dbConnector.clearStatement()
    else:
        #close the cursor
        self.dbConnector.clearStatement()
        self.dbConnector.raiseException("We were unable to create the customer at this time")

def updateCustomer(self, firstNameP=None, lastNameP=None, phoneP=None):
    #fill the method parameters with data from the object if values aren't provided
    #this will ensure the statement has data values, while changing the data provided
    if firstNameP is None:
        firstNameP = self.firstName

    if lastNameP is None:
        lastNameP = self.lastName

    if phoneP is None:
        phoneP = self.phone

    #create the update statement with parameters (i.e., ?) for a prepared statement
    statementString = "UPDATE Customer SET firstName=?, lastName=?,
        phoneNumber=? WHERE id=?"

    #create statement (i.e., cursor), execute the statement, and commit the changes
    self.dbConnector.createStatement(prepared=True)

```

```

self.dbConnector.executeStatement(statementString, (firstNameP, lastNameP, phoneP,
    self.customerId))
self.dbConnector.commit()

#if the execution worked, set the attributes of the class with the data sent to the database
if self.dbConnector.statement.rowcount > 0:
    self.firstName = firstNameP
    self.lastName = lastNameP
    self.phone = phoneP

#close the cursor
self.dbConnector.clearStatement()
else:
    #close the cursor
    self.dbConnector.clearStatement()
    self.dbConnector.raiseException("We were unable to update the customer at this time")

def getCustomerByLastName(self, lastName):
    #create the parameterized prepared statement string using a ? in place of actual data
    statementString = "SELECT id, firstName, lastName, phoneNumber FROM Customer
        WHERE lastName=?"

    #create statement (i.e., cursor), execute the statement, and commit the changes
    self.dbConnector.createStatement(prepared=True)
    self.dbConnector.executeStatement(statementString, (lastName,))

    print("Your search returned the following results:")

    #loop over the results stored in the Cursor object.
    #the results are stored in variables named after the database fields
    for (id, firstName, lastName, phoneNumber) in self.dbConnector.statement.fetchall():
        print(f"{firstName} {lastName} with id {id} can be reached at {phone}")

    #close the cursor
    self.dbConnector.clearStatement()

def deleteCustomer(self):
    #stop the command with an AttributeError if the customerId attribute is not set with a value
    if self.customerId is None:
        raise AttributeError("A customer id must be set to delete a customer")

    #create the delete statement with parameters (i.e., ?) for a prepared statement
    statementString = "DELETE FROM Customer WHERE id=?"

    #create statement (i.e., cursor), execute the statement, and commit the changes
    self.dbConnector.createStatement(prepared=True)
    self.dbConnector.executeStatement(statementString, (self.customerId, ))

```

```

self.dbConnector.commit()

#if rows were deleted, remove class attribute values, otherwise raise and error
if self.dbConnector.statement.rowcount > 0:
    self.customerId = None
    self.firstName = None
    self.lastName = None
    self.phone = None
else:
    self.dbConnector.raiseException("We were unable to delete the customer at this time")

#close the cursor
self.dbConnector.clearStatement()

#when the object is no longer used, close the database connection
def __del__(self):
    self.dbConnector.disconnect()

#instantiate a MySQLConnector object
MySQLdbDriver = MySQLConnector(userP='myUser', passwordP="myPassword",
                                hostP="localhost", databaseP='mis2100example_db')

#instantiate the Customer class and pass in the MySQLConnector object
customer = Customer(MySQLdbDriver)

#multiple database operations can now be called on the Customer object
customer.createCustomer("Jamie", "Doe", "555-555-5554")
customer.updateCustomer(phoneP="555-555-5558")

#destroy the Customer object to close the database connection when you are done with it
del customer

```

After understanding this complex example, you are ready to begin working with database drivers and domain objects to create more loosely coupled systems.

## A Cautionary Security Note

In industry, the connection between programs and databases is more complex than the examples provided in this section. In this chapter, database code was embedded directly into a notebook that runs on your PC, which may or may not be well secured. Although this approach could be fine for home projects, it is not appropriate when a database contains private or proprietary data unless carefully supervised or allowed by the IT department. Directly embedding a database connection into a program like a notebook that a user can easily share with others is not an appropriate approach for a notebook that is to be shared by many users. It would be foolish for Facebook, TikTok, etc. to embed secure database information in their apps because downloadable applications can be hacked in various ways. You don't want users gaining access to database usernames and passwords. Embedding database passwords into

programs is not a particularly secure coding practice for programs that a user can download. Although this approach might be okay if you wish to create programs for your own personal use or under tight supervision of the IT department, it should rarely if ever be used in business settings.

In business settings, a user interface often links to a database indirectly through another program called an application programming interface (API). An API is like a user interface for a computer. API's allow computer programs to communicate with other programs. For example, the pandas-datareader library is a tool that connects to financial APIs and allows you to access the stock data from different data providers. You've already seen the Alpha Vantage API and how to collect stock data from the Alpha Vantage system. APIs sit between your program and the database of the company that owns the data. For example, the Alpha Vantage API connects to their database servers to access their financial data. Then the API returns the data from the database back to your computer through the Python requests library. API's hide details about other company's internal databases. By removing direct database connections, APIs secure databases from abuse. APIs help to limit who can access data and how much data they can access. Databases should be carefully protected, as they often contain highly sensitive or proprietary information.

Even tech-savvy IT or business users don't always require full access to a database. Employees within an organization may be granted limited access to a database through a restricted database user account or through a database tool called a view. Alternatively, some organizations may provide business users with spreadsheets of data exported by IT employees from a database. Each business will develop slightly different approaches to protect information while permitting data to be analyzed for business purposes. Ultimately, organizations should be careful about how they provide data to internal and external users. Pay attention to your organization's data policies and follow them with care.

## Chapter 7 Assignment

Please see the [Chapter Assignments Workbook](#) for the Chapter assignment.

## Section III: Programming for User Interfaces

This section of the text is dedicated to explaining how you can make your computing logic more accessible to business users through graphical user interfaces. Section III includes Chapters 8 and 9.

**Chapter 8** will introduce you to hypertext markup language (HTML) and cascading style sheets (CSS) to improve the visual appeal and usability of your notebooks. You will learn how HTML can be used to structure informative documentation for non-technical users of your notebooks. You will also learn how HTML can be used to structure the outputs of data analytics calculations. CSS is also introduced to teach you how to improve the aesthetics of your notebooks by changing colors, spacing, and other visual elements within your notebooks.

**Chapter 9** will introduce you to how to implement interactive visual interface elements, such as form fields and buttons, to create dynamic experiences for users. You will be introduced to further elements of HTML that support interactive user interfaces. This chapter will also introduce you to the ipywidgets Python library, which makes creating interactive forms within notebooks simple. You will learn basic concepts from event driven programming to help you provide users with interactivity. This chapter will explain how you can integrate your business analytics knowledge with your budding understanding of user interfaces.

After completing this section, you will be able to integrate your foundational understanding of business analytics with knowledge of user interface design to help you create analytics notebooks that can be shared with business users. Learning to make your code readable to non-technical users will enhance the utility of your budding programming skills. Through assignments in the associated workbook, you will learn to write notebooks that can accept input from user interface elements to provide customized output to users.

## Chapter 8:

# Improving the Notebook User Interface

In this text, we explore programming within notebooks. Notebooks are designed to act as an IDE to help you write code. However, they also act as the user interface that you share with users (e.g., your business peers). The results of your code are displayed within the notebook. Notebooks can be published and shared so that others can view and utilize information in your notebooks to make business decisions.

Not all programs are like notebooks. In many cases, the computing logic of a system resides on a server (i.e., a powerful computer). This programming logic is sometimes referred to as the “**backend**” of the system. Python is used in many companies as a backend programming language; the code resides on servers inaccessible to users. Notebooks are one of the few exceptions where Python is directly infused into the user interface.

For programs with a backend, users gain access to backend programming logic and functionality through a graphical user interface that runs on the user’s computing device. Although you write some backend business analytics or machine learning logic into notebooks, you can also connect notebooks to other backend code that resides on servers. For example, you might connect to a database server to access data or to an application server through an application programming interface (API) to borrow from functionality that others have programmed. In Chapter 6, you learned how to use the pandas data-reader library to connect to the Yahoo Finance API to retrieve stock price and volume data.

Almost every time you visit one of your favorite websites, you are accessing graphical user interfaces designed in a markup language called hypertext markup language (HTML). These user interfaces are sometimes called the “**frontend**” of the system. The user’s computing device that displays the frontend is called a **client**. In a traditional program (i.e., not a notebook), the frontend code on the client device calls to the backend programming logic that resides on a server to provide functionality to users through the graphical interface components.

Although we will not examine how to write traditional frontend-backend applications, the topics covered in this chapter can help you create visually appealing notebooks. This chapter will focus on how to use HTML as a frontend markup language to improve the aesthetics and usability of your notebooks. By learning HTML and Python, your programming knowledge will also translate if you ever desire to write or manage the development of a more traditional frontend-backend system.

## Hypertext Markup Language (HTML) for Interface Structure

HTML is a widely used language that produces graphical user interfaces. HTML is the dominant language for writing website interfaces. HTML is not a programming language like Python and Java. Although recent updates to HTML (i.e., HTML5) include some interactive functionality, **HTML mostly lacks control flow logic** like if statements, loops, and try-except statements. It is a **markup language** that tells a browser how to structure graphical elements of the user interface.

Despite HTML's lack of computing logic, frontend interfaces can still execute computing logic for users. Not all programming has to be on the backend. When designing HTML interfaces, frontend computing logic is often provided through a language like Javascript. Javascript can be embedded into HTML structures to create functional and interactive user interfaces that executes on users' computing devices instead of on servers. Frontend Javascript code often connects to backend server code to provide interactivity in traditional systems. Styling can also be added to HTML to improve the aesthetics of the structure provided by HTML. The aesthetics of HTML are usually managed through cascading style sheets (CSS), which beautify HTML interface elements (e.g., add colors, padding, margins, shadowing, etc.).

HTML is also becoming a popular language for writing desktop applications. Traditionally, desktop application interfaces were created using libraries from languages like Python and Java. Many of these libraries produce robotic and outdated looking interfaces. These older interface libraries are often used in an object-oriented fashion, requiring that the interface developer understands programming concepts. Conversely, HTML requires little knowledge of programming. As a markup language, you need to learn about a few HTML elements and you can begin structuring a user interface.

Let's start by introducing you to the fundamentals of HTML markup and then examine how HTML can be used within notebooks.

## HTML elements

Each page in a web interface is an HTML document. Since a notebook runs in a web browser, the notebook itself is an HTML document. HTML documents provide interface structuring through HTML elements. An **element** is simply a visible structure for displaying content (i.e., text, images, videos, etc.) on a user interface. If you visit a website, you will notice that it consists of different sections (ex. header, navigation, main content area, footer, etc.). Each of these sections is displayed by creating HTML **container elements**. These container elements contain other elements that display images, form elements, buttons, links, paragraphs of text, and more.

You create HTML elements by writing **HTML tags**. Each tag has a name and possesses specific default display properties. For example, a paragraph can be created with the "p" tag, which has default display properties that provide white space above and below the paragraph text. Similarly, an unordered bulleted list can be created with the "ul" tag, which has display properties that show list items as indented bullet points. Many different types of HTML tags exist.

Most HTML tags consist of an opening tag and a closing tag. The content that you wish to structure (i.e., text, images, etc.) is placed between the opening and closing tags. Opening and closing tags rely on angle brackets to designate the tag. The closing tag also includes a forward slash along with angle brackets. The basic syntax for an **opening tag** is:

```
<tagName>
```

The basic syntax for a **closing tag** is:

```
</tagName>
```

Content you want to be structured would go between the opening and closing tags like this:

<tag>content to display</tag>

Notice that in the examples above, tags are displayed in blue text and the content within tags is in black text. Like classes, HTML tags can contain attributes/properties as you will see in examples below. Properties are displayed in purple text and the values of the attributes are colored green. This color scheme will be used throughout the text when you see HTML.

## Common HTML elements

For clarification, there is no such thing as a “tagName” or “tag” HTML tag. The examples above exist only to show the syntax for a tag. Actual tags are presented later. When the browser interprets the HTML tags, only the content between the tags is displayed to the user. The angle bracketed tags are not displayed. The tags only designate how the browser should format and display the content. A number of commonly used HTML tags are presented below. There are many other HTML tags for specific purposes. However, those presented below will allow you to create a wide variety of interface structures.

### Commonly Used HTML Tags

The **<html>** tag. Used to tell the browser that all other tags between the opening and closing tag should be interpreted as HTML elements. Only one set of html tags are needed for a specific interface page. Usage:

```
<html> </html>
```

The **<head>** tag. Used as the “brain” of the user interface. The head tag contains the page title that shows on browser tabs, Javascript scripts that provide functionality to the user interface elements, and CSS stylesheets that beautify the HTML elements. The information within the opening and closing head tags are not visible to the user. Only one set of head tags is needed for a specific interface page. The head tag belongs inside the **<html> </html>** tags. Usage:

```
<html>
  <head> </head>
</html>
```

The **<title>** tag. Used to label the entire HTML document. The title tag contains a textual label for the page. This label is displayed in the tab of a browser. Only one set of title tags is needed for a specific interface page. The title tag belongs inside the **<head> </head>** tags. Usage:

```
<head>
  <title> Page Title </title>
</head>
```

The **<body>** tag. Used to contain all of the HTML elements that will be displayed to the user. Anything you want the user to see on the page should go within the opening and closing **<body>** tags. Only one set of body tags is needed for a specific interface page. The body tag belongs inside the **<html> </html>** tags beneath the **<head>** tags. Usage:

```
<html>
  <body> </body>
  <head> </head>
```

```
</html>
```

**The container tags:** `<div>`, `<header>`, `<main>`, `<footer>`, and `<section>`. Used to create large container areas that contain multiple other HTML elements, such as forms, buttons, images, etc. They should be contained inside the body tags. Div tags, short for division, were the original HTML container tag. Recent updates to HTML included more intuitively named container tags, such as header, main, footer, and section tags. They largely perform the same function as div tags, but provide greater clarity to the interface developer. Multiple div, header, main, footer, and section tags can exist on a specific interface page. You can also embed contains within one another. For example, you could have a section container with header, main, and footer containers. Usage:

```
<div> </div>  
or  
<header> </header>  
or  
<main> </main>  
or  
<footer> </footer>  
or  
<section> </section>
```

**The `<nav>` tag.** Used to create a container for navigation buttons. Nav tags are part of the upgrade to HTML5 that included the more intuitively named container tags. Nav tags are similar to div and section tags. Nav tags should be contained inside the `<body>` tags. Although some websites contain one main navigation area, multiple nav tags can be used on a specific interface page. Usage:

```
<nav> </nav>
```

**The `<a>` tag.** Used to create links to other web pages. The “a” tag (a.k.a. the anchor tag) allows you to add a URL as a property of the tag to link to a specific website. The user will see whatever word is contained between the opening and closing “a” tags as a clickable link. The “href” (a.k.a. hypertext reference) property of the “a” tag is set to the website URL you wish to visit. Multiple “a” tags can exist on a specific interface page and should be contained within the body tags. Usage:

```
<a href="https://google.com"> Link Text </a>
```

**The `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>` tags.** Used to create text headings within a document. The h1 tag provides the largest and most prominent heading, followed by h2, h3, and so on. The heading text goes between the opening and closing tags. Multiple heading tags can exist on a specific interface page and should be contained within the body tags. Usage:

```
<h1> Heading Text </h1>
```

**The `<p>` tag.** Used to create paragraphs within a document. The paragraph text goes between the opening and closing tags. Multiple “p” tags can exist on a specific interface page and should be contained within the body tags. Usage:

```
<p> Paragraph Text </p>
```

**The `<ul>`, `<ol>`, and `<li>` tags.** Used to create unordered bulleted lists or alpha-numerically ordered lists of items. The ul tags create unordered lists. The ol tags create ordered lists. The li tags are used within the ul and ol tags to create individual list items. You will need one li tag for each item you want

to be displayed in the list. The text of each list item goes within the opening and closing li tags. Multiple ul and ol tags can exist on a specific interface page and should be contained within the body tags. Usage:

```
<ul>
    <li> List Item 1 Text </li>
    <li> List Item 2 Text </li>
    <li> List Item 3 Text </li>
</ul>
```

The **<img> tag**. Used to display images within a document. The image tag does not have a closing tag. Only an opening image tag is required. The image tag has a “src” property that allows you to enter the file location of the image you want to display. Multiple img tags can exist on a specific interface page and should be contained within the body tags. Usage:

```

```

The **<br> tag**. Used to create a line break in the document. The br tag does not have a closing tag. Only an opening image tag is required. Multiple br tags can exist on a specific interface page and should be contained within the body tags. Usage:

```
<br>
```

The **<!-- --> tag**. Used to add comments to your HTML code that are not visible to the user. Provides feedback to other developers who may need to edit an HTML page document. The comment goes within the <!-- and the --> characters. Multiple comments can be added to an HTML document.

Usage:

```
<!-- Comment Text Goes Here -->
```

## HTML table elements

It is possible to create tables within HTML pages, which is useful for many financial applications. Many financial reports rely on tables to present data in a user-friendly format. HTML tables are included in a separate section because tables consist of several different tags structured in a specific way.

To specify a table, you will use the `<table></table>` tags. Tables consist of two main regions, the table head and the table body. The table head is represented by the `<thead></thead>` tags and the table body by the `<tbody></tbody>` tags. Within the `<thead></thead>` and `<tbody></tbody>` tags, you will add table row tags `<tr></tr>`. Within the `<thead></thead>` tags, you will normally have only one table row. Within the `<tbody></tbody>` tags, you will have as many rows as you have data entries to display. The `<tr></tr>` in the table head represents the row with the labels for each column in the table. Columns are represented by `<th></th>` or `<td></td>` tags within `<tr></tr>` tags. The `<th></th>` tags represent headings for each column in the table. The `<td></td>` tags represent the table data (i.e., td) for each column in each row. This written description may have been confusing because of the several elements of a table. Let's examine an example.

As with other display tags, `<table></table>` tags belong in the `<body></body>` of a page.

## Basic HTML Table Structure

The tags below create a table with three columns. The first row (i.e., in the <thead>) will show headings for each of the three columns. The <tbody> consists of three rows with three columns to match the structure of the table head.

```
<table>
  <thead>
    <tr>
      <th>Column 1 Heading</th>
      <th>Column 2 Heading</th>
      <th>Column 3 Heading</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Row 1 Column 1 Data</td>
      <td>Row 1 Column 2 Data</td>
      <td>Row 1 Column 3 Data</td>
    </tr>
    <tr>
      <td>Row 2 Column 1 Data</td>
      <td>Row 2 Column 2 Data</td>
      <td>Row 2 Column 3 Data</td>
    </tr>
    <tr>
      <td>Row 3 Column 1 Data</td>
      <td>Row 3 Column 2 Data</td>
      <td>Row 3 Column 3 Data</td>
    </tr>
  </tbody>
</table>
```

The basic structure presented above will help you create readable financial output for users.

## HTML element properties

As mentioned briefly, HTML elements can have properties. HTML elements are like objects in object-oriented programming. Each element can have many properties/attributes that describe features of the element. In the examples, you saw the “src” property of the img tag and the “href” property of the “a” tag. Many HTML elements support properties. Properties are set on HTML elements within the opening tag. Within the angle brackets of the opening tag, multiple properties can be set with a space between each property. The properties of a tag only affect that certain HTML tag, just like each object instantiated from a class is independent of other instantiated objects. The syntax for setting properties on an HTML element is as follows:

```
<tag property1="value1" property2="value2" propertyN="valueN"> Tag Contents </tag>
```

Examples of properties that can be used on HTML tags include:

- `id` - sets an id for the HTML element. It should be a unique id.
- `name` - sets a name for the HTML element.
- `type` - sets the type of HTML elements; each type behaves in a different way.
- `style` - sets a CSS style for the HTML element.
- `href` - sets a URL location.
- `src` - sets the file location of a resource, such as an image.
- `height` - sets the height of an element. This can also be done through CSS styling.
- `width` - sets the width of an element. This can also be done through CSS styling.

## HTML page structure

HTML tags are embedded within one another to create complex visual structures to display to the user. The basic structure of any web page is displayed below. Notice how the tags are embedded within one another.

### Basic Structure of an HTML Document

The HTML code below would create a single web page. If opened in a browser, the browser tab would display “Page Title”. Because the body is empty, the browser would show a blank, white page. Notice that all of the tags are contained within the `<html></html>` tags. The title is contained within the head tags. The head tag is on top of the body tag, just like our own head sits on top of our bodies.

```
<html>
  <head>
    <title> Page Title </title>
  </head>
  <body>
    <!-- Everything you want the user to see goes here --&gt;
  &lt;/body&gt;
&lt;/html&gt;</pre>
```

The example above can be expanded to produce customized interfaces. Again, all of the content you want to be visible on a page should be contained within the opening and closing body tags. The example below shows a more complex embedding of tags that creates a structured page.

### Simple HTML Page

The HTML code below would create a single web page. Look at the code. What would the user see?

```
<html>
  <head>
    <title> Page Title </title>
  </head>
  <body>
```

```

<header>
    
</header>
<nav>
    <a href="home.html">Home</a> |
    <a href="about.html">About Us</a> |
    <a href="contact.html">Contact Us</a>
</nav>
<main>
    <section>
        <h1>Welcome to My Site</h1>
        <p>This is a paragraph about me!</p>
        <p>This is another paragraph about me!!</p>
    </section>
    <section>
        <h2>My Education</h2>
        <p>I attended the following universities:</p>
        <ul>
            <li>University of Utah</li>
            <li>Brigham Young University</li>
            <li>University of North Carolina at Greensboro</li>
        </ul>
    </section>
</main>
<footer>
    Copyright 2022. All Rights Reserved.
</footer>
</body>
</html>

```

Most HTML tags, particularly those contained within the body tags, display as stacked blocks. That is, the HTML elements stack on top of each other visually. If you ran the example code in a browser, it would show a logo at the top of the page. Underneath the logo would be three links labeled: Home, About Us, and Contact Us. Under the links, you would see the “Welcome to My Site” heading with two paragraphs under that heading. You would then see a slightly smaller heading that says “My Education”. Under the smaller heading, you would see a paragraph, and under that a bulleted list with the names of three universities. Each item in the list would be stacked on one another like blocks. Finally, the last line would display “Copyright 2022. All Rights Reserved”. By default, most HTML tags stack on top of each other in this manner. Armed with this knowledge, you can begin to develop important structures within an interface.

With an understanding of the HTML elements available, you are prepared to start designing interfaces. HTML helps to structure the interface content. However, to further improve the aesthetics of the interface, you will need a basic understanding of cascading style sheets (CSS).

# Cascading Style Sheets (CSS) for Aesthetic Designs

HTML creates structured interfaces, but not aesthetically pleasing interfaces. To create aesthetically pleasing designs, like those you encounter on the web, you must style HTML structures. HTML is generally styled through cascading style sheets (CSS).

To get a feel for how CSS can affect the look and layout of HTML structures, visit this web page: [https://www.w3schools.com/css/css\\_intro.asp](https://www.w3schools.com/css/css_intro.asp). Under the CSS Demo section of the website, click the links for “Stylesheet 1”, “Stylesheet 2”, “Stylesheet 3”, etc. The HTML content of each of the links is exactly the same. However, through styling, each page is made to look completely different. The “No Stylesheet” link will show you what the page looks like with only HTML. As you can see, it is rather bland and boring. CSS adds color and interactivity (e.g., button colors change when hovered over).

## **CSS rules and declarations**

CSS works by applying display **rules** that consist of CSS **declarations** to one or more HTML elements. For example, you might want to change the background color of a `<header>` tag to be black and the text within the `<header>` to be white. These styling changes together represent a rule for `<header>` tags. Each of these styling changes (i.e., background color = black and text color = white) can be represented as a CSS declaration. **CSS declarations** are formed as key-value pairs similar to dictionaries in Python.

To use CSS declarations, you simply need to know the name of the property you wish to change (i.e., `background-color` for background color and `color` for text color), and the acceptable values each property can hold. Each CSS declaration is ended by a semi-colon. The set of declarations, which make up the CSS rule, are contained within curly brackets. The curly brackets and semicolons are reminiscent of the Java programming language. For example, the rule that would create a black background with white text for the `<header>` tag would be written with the following syntax:

```
header { background-color: black; color: white; }
```

Notice that the name of the tag you wish to change comes first. This is called the rule **selector**. Then, within curly brackets a set of declarations are provided. For readability, sometimes CSS declarations will be declared on different lines. Browsers ignore the indentations and line breaks. For example:

```
header {  
    background-color: black;  
    color: white;  
}
```

Notice in the examples above that the selectors are displayed in **golden** text, the selector properties are in **purple** text, and the property values are colored **green**. This convention will be used throughout this text when presenting CSS code.

The table below lists a number of common and useful CSS declarations. For a more complete list of declarations with usage examples, visit the following site: <https://www.w3schools.com/cssref/>.

CSS property	Purpose	Acceptable Values
background-color	Change the background color of an HTML element. The default color for most elements is white. Colors can be set with textual, hexadecimal, red-green-blue (RGB), or hue-saturation-lightness (HSL) values.	Color names: ex. <code>white</code> , <code>black</code> , <code>red</code> or Hexadecimal values: ex. <code>#FF00EE</code> or RGB values: <code>rgb(255, 0, 100)</code> or HSL values: <code>hsl(0, 50%, 25%)</code>
background-image	Add a background image to an HTML element. The content of the element will be layered on top of the background image.	Must specify a URL to the image: <code>url("path/to/image.png")</code>
border	Add a border around an HTML element. Borders consist of a width (e.g., pixels), style (i.e., solid, dotted, etc.), and color. Only the style value is required.	Width, style, and color can be added: <code>border: 5px solid red;</code> <code>border: 1px dotted #FF00FF;</code> <code>border: double;</code>
box-shadow	Add a shadow to an HTML element. Box-shadow consists of required horizontal and vertical offsets to position the shadow. Positive horizontal offset values place the shadow to the right of the element and negative values to the left. Positive vertical offset values place the shadow below the element and negative values on top. Optional properties, such as blur and spread, can also be added.	H-offset, v-offset, blur, spread, color, and other values can be added in that order: <code>box-shadow: 3px 5px;</code> <code>box-shadow: 3px 5px 10px;</code> <code>box-shadow: 3px 5px 10px 5px;</code> <code>box-shadow: 3px 5px 10px 5px grey;</code>
color	Change the color of text within an HTML element. Colors can be set with textual, hexadecimal, red-green-blue (RGB), or hue-saturation-lightness (HSL) values.	Color names: ex. <code>white</code> , <code>black</code> , <code>red</code> or Hexadecimal values: ex. <code>#FF00EE</code> or RGB values: <code>rgb(255, 0, 100)</code> or HSL values: <code>hsl(0, 50%, 25%)</code>
cursor	Change the style of the mouse cursor when the mouse cursor is over an HTML element, such as changing from a default arrow to a pointer hand or to a crosshair.	Cursor type: ex. <code>default</code> , <code>crosshair</code> , <code>pointer</code> , <code>progress</code> , <code>text</code> , <code>zoom-in</code> , <code>zoom-out</code>
display	Change how an HTML element displays. Many HTML elements display as blocks that stack on top of one another (i.e., block display). Other elements can display inline,	Display type: ex. <code>block</code> , <code>inline</code> , <code>grid</code> , <code>flex</code>

	which places elements side by side. Inline styling can change block elements to inline elements. Other advanced display options include grid and flex.	
float	Change how block elements are positioned. By default, block elements stack on top of one another. Floating left or right moves the blocks side by side instead of stacking.	Float type: <code>left, right</code>
font	Change the size and look of the text within an HTML element. Several properties can be specified through the font declaration, such as font-style (e.g., italics), font-weight (e.g., text boldness), font-size (e.g., line height), and font-family (e.g., Arial, Times New Roman).	Values for various font properties in the order font-style, font-variant, font-weight, font-size, font-family. <code>font: italic bold 13px Arial;</code> <code>font: 16px "Times New Roman";</code>
height	Changes the height of an HTML element.	Values in pixels, cm, percentages, etc. For example, <code>50px, 30%</code>
margin	Changes the spacing between an HTML element and surrounding elements on the top, right, bottom, and left of the element, in that order.	Values in pixels, cm, percentages, etc. <code>margin: 5px 10px 15px 10px;</code> <code>margin: 5px;</code>
opacity	Changes the transparency level of an HTML element.	Values between <code>0.0</code> and <code>1.0</code> . Values closer to 0 are more transparent and values closer to 1 are more opaque.
padding	Changes the spacing between the border of an HTML element and the contents of the element. Padding values specified in the order of the top, right, bottom, and left sides of the element.	Values in pixels, cm, percentages, etc. <code>padding: 5px 10px 15px 10px;</code> <code>padding: 5px;</code>
text-align	Changes the alignment of text within an HTML element.	Alignment values, such as: <code>left, right, center, or justify.</code>
width	Changes the width of an HTML element.	Values in pixels, cm, percentages, etc. For example, <code>50px, 30%</code>
z-index	Changes how an HTML element is layered in front of or behind other HTML elements.	Positive and negative integers. Higher values place an element in front of elements with lower values.

With these properties, you can create a variety of declarations and rules that will make your pages look more interesting and aesthetically pleasing.

## CSS *selectors*

CSS selectors provide different ways to apply CSS rules. CSS rules can be applied through **tag selectors, id selectors, and class selectors**.

### Tag *selectors*

**Tag selectors** apply CSS rules to all instances of a particular tag within an HTML document. Tag selectors help to ensure that all tags of a certain type have consistent formatting and styling. For example, if you want every paragraph in your HTML document to be styled a certain way, you could apply the set of styling declarations to the “p” tag.

The code below shows how tag selectors can be used for styling with examples of the “p” and “h2” tags. The header tag example that was provided earlier in the chapter is also an example of a tag selector. If the code below were implemented in an HTML document, all “p” tags would have a dark gray text color with a slightly larger bottom margin. All “h2” heading tags would display in 18 pixel tall, black, Arial font. And all header tags in the document would have a black background with white text.

#### Examples of CSS Tag Selectors

```
p {  
    color: #222222;  
    margin: 0px 0px 10px 0px;  
}  
  
h2 {  
    font: 18px Arial;  
    color: black;  
}  
  
header {  
    background-color: black;  
    color: white;  
}  
  
##### Example of the Associated HTML Code #####  
  
<html>  
<head>  
    <title>Sample Page</title>  
</head>  
<body>  
    <header>  
          
    </header>  
    <section>
```

```
<h2>Section 1 Heading</h2>
<p>The text for section 1.</p>
</section>
<section>
    <h2>Section 2 Heading</h2>
    <p>The text for section 2.</p>
</section>
</body>
</html>
```

### ***Id selectors***

Some HTML elements require an “id” attribute that uniquely identifies a particular HTML tag. Within a given HTML document, every id value should be unique. However, you can repeat the same id values across different HTML documents. In terms of styling, **id selectors** can be useful for creating styles for unique elements that exist across multiple pages.

For example, on each individual web page (i.e. HTML document), there is usually one header area that represents the masthead of the website (i.e., typically where you find the site’s logo) and one main navigation area, but these header and main navigation areas are often repeated across multiple web pages. In such cases, an id attribute can be added to the HTML tag that represents the unique area, and styling can be applied to the id selector. Id selectors are designated by the # symbol prefixed to the name of an id.

In the code below, three id selectors are shown. The example below assumes that the HTML consists of a tag with an id attribute set to “mainHeader”, another tag with an id set to “mainNavigation”, and yet another set to “leftSidebar”. Although it would make sense for the “mainHeader” id attribute to exist within a `<header>` tag, the “mainNavigation” id attribute within a `<nav>` tag, and the “leftSidebar” id attribute within a `<section>` or `<aside>` tag, this doesn’t have to be the case. The CSS rule will be applied to whatever tag carries the respective id value.

### **Examples of CSS Id Selectors**

```
#mainHeader {
    background-color: black;
    padding: 10px 10px 10px 10px;
}

#mainNavigation {
    background-color: red;
    color: white;
}

#leftSidebar {
    float: left;
```

```
        background-color: black;  
        color: white;  
    }
```

```
#####
Example of the Associated HTML Code #####
#####
```

```
<html>  
<head>  
    <title>Sample Page</title>  
</head>  
<body>  
    <header id="mainHeader">  
          
    </header>  
    <nav id="mainNavigation">  
        <a href="home.html">Home</a> | <a href="about.html">About</a>  
    </nav>  
    <section id="leftSidebar">  
        <p>Sidebar content would go here.</p>  
    </section>  
    <section>  
        <p>The main content of the page would go here.</p>  
    </section>  
</body>  
</html>
```

## Class selectors

**Class selectors** are used to create styling that can be applied to as many HTML elements as you wish to apply the rule to. Class selectors can be applied across different HTML elements (e.g., applied to a “p” tag, “h2” tag, and “header” tag). They can also be applied to some, but not all tags of a certain tag type (e.g., all “p” tags except for the first “p” tag of each section on the page). Class selectors are the most versatile of the selectors. Class selectors are designated with the “.” symbol prefixed to the name of the selector.

If the term “class” sounds familiar, it is. Class selectors are based on the classes that exist in object-oriented programming languages. If you recall from previous chapters, classes allow you to create repeatable logic and structure that can be instantiated many times to create objects from that class. Class selectors serve the same purpose in CSS. First, a CSS class is created with styling declarations with a name given to the class by the designer. Then, the class is “instantiated” by applying it to particular HTML tags using the “class” attribute of the HTML elements.

The code below shows the creation of three class selectors and application of the classes to HTML tags.

### Examples of CSS Class Selectors

```

.redItalicText {
    color: red;
    font: italic 14px;
}

.blackBackground {
    background-color: black;
    color: white;
}

.greenBorder {
    border: 1px solid green;
    padding: 10px;
}

#####
Example of the Associated HTML Code #####
#####
```

```

<html>
<head>
    <title>Sample Page</title>
</head>
<body>
    <header class="blackBackground">
        
    </header>
    <section class="greenBorder">
        <p class="blackBackground">Sidebar content would go here.</p>
        <p class="redItalicText">Sidebar content would go here.</p>
    </section>
    <section class="redItalicText">
        <p class="greenBorder">The main content of the page would go here.</p>
        <p>A paragraph without a class.</p>
    </section>
</body>
</html>
```

## Selectors with pseudo-classes

A **pseudo-class** is often used to change the styling of an HTML element when the state of the element changes. For example, users on the web can hover over and click buttons. Different styling can be applied to elements when they are hovered over, clicked, or otherwise interacted with. Pseudo-classes can be applied to tag, id, and class selectors. Many different pseudo-classes can be applied to the same selector.

To create a pseudo-class, you must first create the selector that it should be applied to. This selector should contain styling rules for the default state of the HTML tag. Then, a new selector is created with the same name and the associated pseudo-class you wish to style for (e.g., hover, active, etc.). The pseudo-class selector contains styling declarations for the interaction effect. The pseudo-class selector only needs styling changes that change when an interaction takes place; it will inherit any styling from the default selector that doesn't need to change. The basic syntax for a pseudo-class is:

```
selector {}  
selector:pseudo-class {}
```

Although many pseudo-classes exist, we will explore two commonly used pseudo-classes: hover and active. The **hover pseudo-class** allows you to change styling when a mouse cursor hovers over an HTML element. The hover pseudo-class is often used to signal that an element is interactive, such as a button. When you see a button on the web that changes color when you hover over it, that is a pseudo-class in action. The **active pseudo-class** allows you to change styling when an element is clicked on. The active pseudo-class can be used to signal an action has been taken, such as a button press. For example, you might want to remove the shadowing under a button when it is clicked to make it look like the button is being depressed.

### Examples of CSS Pseudo-class Selectors

```
button {  
    background-color: #000000;  
    color: white;  
}  
  
button:hover {  
    background-color: #0000FF;  
}  
  
.cancelButton {  
    background-color: red;  
    color: white;  
    box-shadow: 5px 5px 10px #000000;  
}  
  
.cancelButton:active {  
    box-shadow: 0px 0px 0px #000000;  
}  
  
##### Example of the Associated HTML Code #####  
  
<html>  
<head>  
    <title>Sample Page</title>  
</head>  
<body>
```

```
<section>
    <button>Submit</button>
    <button class="cancelButton">Cancel</button>
</section>
</body>
</html>
```

With an understanding of tag, id, and class selectors and pseudo-classes, you can begin to develop interesting user interfaces. Importantly, you can mix and match different types of selectors (e.g., tag, id, and class selectors) into a single HTML document.

## Implementing CSS Styles

You now understand the basics of styling with CSS. But how do you actually integrate CSS rules into an HTML document? Three different approaches exist to integrate CSS code into an HTML document: internal style sheets, external style sheets, and inline styles.

**Internal style sheets** are contained within the HTML document itself. CSS rules can be written directly into an HTML document within the `<head>` tag of the document. Within the head tags, you simply need to create opening and closing `<style>` tags. Between the style tags, you can write CSS syntax as in the previous examples. The code example below shows the use of tag, id, and class selectors with an internal stylesheet in an HTML document.

### Integrating CSS into an HTML Document with Internal Style Sheets

```
<html>
<head>
    <style>
        p { color: #222222; margin: 0px 0px 10px 0px; }
        #mainHeader { background-color: black; padding: 10px 10px 10px 10px; }
        .cancelButton { background-color: gray; box-shadow: 5px 5px 10px #000000; }
        .cancelButton:active { box-shadow: 0px 0px 0px #000000; }
    </style>
</head>
<body>
    Content
</body>
</html>
```

**External style sheets** are contained within a separate file with the file extension `.css`. External style sheets are then imported into an HTML document within the `<head>` tag. In this way, one CSS file can be used across multiple HTML documents. Notice in the example below that the CSS code is contained in a file named `style.css`. The `style.css` file is then referenced in the head of the HTML document by using the `<link>` tag.

## Integrating CSS into an HTML Document with External Style Sheets

This code would be in a file named style.css:

```
p { color: #222222; margin: 0px 0px 10px 0px; }
#mainHeader { background-color: black; padding: 10px 10px 10px 10px; }
.cancelButton { background-color: gray; box-shadow: 5px 5px 10px #000000; }
.cancelButton:active { box-shadow: 0px 0px 0px #000000; }
```

This code would be in a file named page1.html:

```
<html>
<head>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    Content
</body>
</html>
```

**Inline styles** offer a third way to integrate CSS into an HTML document. Inline styles don't require selectors. Instead, the CSS declarations are embedded directly into individual HTML tags. CSS declarations are entered into the tags through the style attribute of the tag. The code below shows how CSS can be implemented inline into individual HTML tags.

Except in certain instances, inline styles should be used sparingly. They are more difficult to change than internal and external stylesheets. For example, consider that you were designing an interface for 100 different HTML pages. Assume that the pages had an average of 5 “p” tags per page. If you were to style each of the “p” tags with an inline style, you would need to copy/write 500 rules (i.e., a rule in each “p” tag on each page). If you were to write the same rules in internal style sheets, you would need to write 100 rules (i.e., copying the rules across the 100 pages) into the internal stylesheet in the <head> tags of each page. If you were to write the same rules in an external style sheet, you would only need to write the rule once and import the sheet into each of the 100 pages. Now imagine that you needed to update the rules once per year or more. External style sheets offer the most efficient method to implement and manage CSS, followed by internal style sheets, and then inline styles.

## Integrating CSS into HTML Elements with Inline Styling

Note the CSS is included in the style attributes of the HTML tags:

```
<html>
<head>
    <title>Sample Page</title>
</head>
<body>
    <header style="background-color: black; padding: 10px 10px 10px 10px;">Content</header>
    <p style="color: #222222; margin: 0px 0px 10px 0px;">Content</p>
    <button style="background-color: gray; box-shadow: 5px 5px 10px #000000;">Save</button>
</body>
```

```
</html>
```

Notebooks offer one exception to the use of external, internal, and inline style sheets. If you have a standard style sheet that you want to apply to multiple notebooks within your organization, external style sheets are still the most efficient option. However, if you are designing a notebook that will only be used for a single business problem that needs to look professional, then internal style sheets or even inline styles can be just as efficient as external stylesheets.

## Using HTML and CSS in Notebooks

You now possess foundational knowledge about HTML and CSS. What you have learned has prepared you to write simple web pages. However, it has not fully prepared you to structure and style notebooks. You cannot write HTML and CSS into every notebook cell and expect it to work correctly. HTML and CSS can be implemented in notebooks, but in specific and sometimes different ways depending on the type of cell. We explore a few methods to embed HTML and CSS in notebooks below. Let's start by examining ways to create internal and external style sheets in notebooks. We will then move to writing HTML code in notebooks.

### *Implementing internal style sheets in notebooks*

Inline styles can be used in any of your HTML code. However, it is often better to use internal or external stylesheets. Internal and external style sheets are easier to edit and manage than inline styles.

In notebooks, it is often easiest to either import an external style sheet or write an internal stylesheet into one of the first cells of the notebook. The first cell of a notebook is typically used to import Python libraries. The second code cell could be used to import or create your CSS style sheets. Similar to importing Python libraries, once an external stylesheet or internal style sheet is created, it will apply to all other cells. That is, the same CSS style sheet can be used to style the output of all code cells and the content of all markdown cells. Where possible, rely on internal and external style sheets.

Before writing internal style sheets into a coding cell, you must first import an iPython library that allows you to implement HTML and CSS into code cells. The import statement you need to include in your notebook is:

```
from IPython.core.display import HTML
```

Once the HTML display library is added to the notebook, you can create an inline style within `<style>` tags in one of the first few notebook cells. You want to add it to an earlier code cell so that the styling is applied to the rest of the cells when you begin running each cell. Typically, `<style>` tags are included in the `<head>` tag of an HTML document. However, in the case of notebooks, the `<style>` tag can be implemented into a cell, which is not in the `<head>` tags.

The HTML display library offers two main ways to implement HTML into a code cell. The first converts the entire cell into an HTML interpreter. This is accomplished through the use of magic commands. **Magic commands** allow you to perform complex behaviors with a simple syntax. They are called “magic” because they hide the logic behind a simple interface (i.e., the % symbol). In notebooks, magic

commands are denoted with the % symbol. To convert an entire cell into an HTML/CSS interpreter, first ensure that you have imported the HTML display library. Then, use the following magic command to turn the entire cell into an HTML interpreter:

```
%%html
```

The magic command should be entered into the first line of the cell. Then every line after the magic command can be written as HTML or CSS with <style> tags. In many ways, the %%html magic command converts a code cell into a markdown cell. However, the magic command provides the added bonus of interpreting CSS internal style sheets. Markdown cells don't support internal style sheets in this way. The code below shows how the %%html magic command can be used to create an internal style sheet. Again, you will want your style sheet to be in one of the first code cells so that it is applied to all later cells as they run.

#### Integrating a Internal Style Sheet into Code Cells with the %%html Magic Command

The following would be added to one of the first cells:

```
%%html
<style>
    body { background-color: #000000; }
    .sectionHeader {background-color: black; color: white; padding: 10px; font: 24px Arial;}
    .sectionBody {background-color: white; color: black; padding: 10px; font: 14px Arial;}
</style>
```

Internal style sheets can also be implemented without the %%html magic command by using the HTML() method of the html display library mentioned previously. To do this, your internal style sheet must be contained within a string. Implementing a style sheet on a single line string makes the CSS code hard to read and edit. As such, you will want to create a **multi-line string** to make writing and editing the CSS easier. In Python, **multi-line strings** can be created using three quotation marks instead of one quotation mark. For example:

```
"""
<section>
    <p>A multiline string starts with three quotation marks.</p>
    <p>I can have multiple lines without an error occurring.</p>
    <p>To end the multiline string, three more quotation marks are needed.</p>
</section>
"""
```

HTML and CSS strings can be passed through the HTML() method of the HTML display library so that your HTML and CSS code is interpreted correctly by the browser. The code below shows how the HTML() method can be used to create an internal style sheet. If the code were run, the styles in the style sheet string would be applied to all of the cells in the notebook.

#### Integrating a Internal Style Sheet into Code Cells with the HTML display Library

The following library would need to be imported first:

```
from IPython.core.display import HTML
```

After importing the library, the following code would be written in a code cell.

#Notice the three quotes surrounding the CSS code.

```
css = """  
<style>  
    body { background-color: #000000; }  
    .sectionHeader {background-color: black; color: white; padding: 10px; font: 24px Arial;}  
    .sectionBody {background-color: white; color: black; padding: 10px; font: 14px Arial;}  
</style>  
"""
```

#The CSS string is then included as an argument to the HTML() method of the display library

```
HTML(css)
```

## ***Implementing external style sheets in notebooks***

If you have an external style sheet hosted on a web server, you can also import that style sheet into a code cell to be applied to the rest of your notebook. Some organizations have style sheets hosted on web servers to create a consistent, organizationally-approved style guide for notebooks and other interfaces. To utilize an external style sheet, you would need to know the URL of the style sheet. For example, at the time of this writing, one of the style sheets used at Facebook can be located at the URL:

[https://static.xx.fbcdn.net/rsrc.php/v3/yO/l/0.cross/Byksuglcc0gsHUpitHChBN.css?\\_nc\\_x=GXnsdV1Sj8R](https://static.xx.fbcdn.net/rsrc.php/v3/yO/l/0.cross/Byksuglcc0gsHUpitHChBN.css?_nc_x=GXnsdV1Sj8R)

To import an external style sheet hosted on a web server, you will need two Python libraries. The first is the HTML display library demonstrated earlier. The second is a library to retrieve the contents of a URL. Many URL content retrieval libraries exist. We will use the urllib library to read in external style sheets. The code to import the specific urllib tools that you will need is:

```
from urllib.request import urlopen
```

When importing a web-hosted external style sheet into a notebook, **you will need to convert the external stylesheet into an internal style sheet**. First, the urlopen module of the urllib library is used to read the contents of the web-hosted style sheet. Second, the results of urlopen must be decoded to a format that a web browser can read (i.e., UTF-8). We will not cover encoding techniques in this text. If you are interested in learning how different programs encode and decode content, a more traditional computer science textbook or Google can help. We will be decoding the results of the urlopen into UTF-8 format. Third, the decoded results need to be wrapped in <style> tags to turn the external style sheet into an internal style sheet. Last, the decoded result is passed through the HTML display library so it can be interpreted as CSS.

When the cell is run, the web-hosted style sheet will be imported into the notebook and interpreted as CSS. The rules in the style sheet will then be applied to all other cells in the notebook. The code below shows how to import a web-hosted external style sheet.

### Integrating a Web-hosted External Style Sheet into Code Cells with urllib

The following libraries would need to be imported first:

```
from IPython.core.display import HTML
from urllib.request import urlopen
```

After importing these libraries, the following code would be written in another cell.

```
#uses the urlopen() method of urllib.request to connect to a URL
#the read() method is then used to get the content from the URL
css = urlopen("https://style.com/style.css").read()

#the contents of the read() method are then decoded using the decode() method
decodedCSS = css.decode("utf-8")

#the CSS style sheet is wrapped in <style> tags to create an internal style sheet
finalCSS = f"<style>{decodedCSS}</style>

#the decoded CSS style sheet is then interpreted by the HTML display library as CSS
HTML(finalCSS)
```

You can also implement external style sheets stored on your own computer. The process is similar to opening a style sheet from a web server. You simply use the built-in open() method in Python to read in a CSS file stored on your computer. For this to work, you need to know the full path to the file on your file system. For example:

C:/Users/user/myWebsite/style.css

The code below shows how an external style sheet stored on your computer can be added to a notebook. Notice that the process is a little simpler than reading a web-hosted style sheet. You do not need to decode the results.

### Integrating an External Style Sheet from your Computer into Code Cells with open()

The following library would need to be imported first:

```
from IPython.core.display import HTML
```

After importing the library, the following code would be written in another cell.

```
#uses the open() method in Python to connect to a file on your computer. The "r" is for read mode.
```

```
#the read() method is then used to get the content from the file.  
css = open("C:/Users/user/myWebsite/style.css", "r").read()  
  
#the CSS style sheet is wrapped in <style> tags to create an internal style sheet  
finalCSS = f"<style>{css}</style>"  
  
#the CSS style sheet is then interpreted by the HTML display library as CSS  
HTML(finalCSS)
```

## Implementing HTML in markdown cells

Now that you know how to implement style sheets into a notebook, let's explore how HTML can be included into notebooks. **The simplest way to embed HTML in a notebook is to write HTML directly into a markdown cell.**

If you will recall from earlier chapters, notebook cells can be set to either “Code” or “Markdown” mode. We have primarily explored code cells in previous chapters. **Markdown cells** are used within notebooks to provide textual descriptions and notes for yourself or others. For example, you might want to document why you made certain programming or analysis choices in your notebook. Or, you might want to explain the results of a statistical analysis in terms that your audience can understand.

Rather than simply adding comments into the code itself, which would be unfriendly for non-technical users, you can create a markdown cell and provide meaningful descriptions for yourself or other users. Markdown cells are intended to be more readable and accessible to those who might not understand or want to read code. However, you cannot include code in these cells. Markdown cells are designed for text and HTML output. Some use Markdown cells in presentations to explain analyses and results similar to the way you would utilize a slideshow presentation.

You can write HTML code into a markdown cell as if it were an IDE built for HTML code. However, you will NOT need the <html>, <head>, or <body> tags. Notebooks run inside of a browser. As such, they are themselves built with HTML and CSS. That means each notebook already contains <html>, <head>, and <body> tags to display the notebook interface in the browser. Within each cell, you simply need to write the HTML that would be contained within the <body> tags. Notice in the example code below that the html, head, and body tags are not included. The code below could only be entered in a markdown cell. You would receive an error if you tried to write HTML directly into a code cell.

Even though the Markdown cell does not contain code, you still need to click the “run” button in Jupyter Notebook/Lab to convert your HTML code into the actual interface output. **To edit your code after you run a Markdown cell**, simply double click the Markdown cell and it will return to the HTML code view where you can edit the HTML code.

### Integrating HTML and CSS into a Markdown Cell of a Notebook

Notice that the CSS in the “p” tag is embedded through inline styling using the style property. Although the <button> tag below references a CSS class, the CSS rules for the submitButton class

cannot be added to the markdown cell. Instead, the submitButton class rule would be written in an earlier code cell using the methods of adding a style sheet to a notebook mentioned previously. The class would be applied to the button when the cell with the code below is run.

```
<header>Markdown Cell Header</header>
<section>
    <p style="color: #222222; margin: 0px 0px 10px 0px;">Content of the cell.</p>
        <button class="submitButton">Submit</button>
</section>
```

CSS can only be directly implemented into a markdown cell through inline styles. Internal and external style sheets do not work properly in a markdown cell. To style the HTML within a markdown cell, using either an internal or external style sheet, you must first include the internal or external style sheet in a code cell.

## ***Implementing HTML in code cells***

HTML can be implemented into notebook code cells in two different ways. First, a code cell can be converted into a HTML interpreter by using the %%html magic command. This is the same approach used to create an internal style sheet in a notebook. This method does not allow you to intermingle HTML into your code output. Instead, this method turns the code cell into a markdown cell. In most cases, it is easier to simply write raw HTML code into a markdown cell than to convert a code cell using the %%html magic command. The one exception is when you create an internal style sheet. Markdown cells readily support HTML without magic commands, except for CSS style sheets.

Second, HTML can be mingled with Python code in a code cell to create stylized output for any calculations you perform. Often, you will want to format the output of your code cells. HTML, in conjunction with CSS, can be added to code cells to structure and style the output of the cells. The approach is similar to including style sheets in a code cell using multi-line strings. You will use multi-line strings and the HTML() method of the iPython display library to display structured and stylized cell outputs.

### **Integrating HTML and CSS into a Code Cell of a Notebook**

The code below assumes an internal/external style sheet exists with a CSS class called submitButton in an earlier cell. The following library would need to be imported first:

```
from IPython.core.display import HTML
```

Assume the following CSS style sheet were included in an earlier cell:

```
<style>
    .subHeader {background-color: black; color: white; padding: 10px; font: 24px Arial;}
    .contentSection {background-color: #EEEEEE; padding: 10px;}
    .label {font-weight: bold;}
</style>
```

After importing the library and CSS, the code could be written. Notice the three quotes used to create a multiline string with code output embedded in the string.

```
class Customer:  
    #constructor and getter/setter methods would go here  
  
    def showCustomerDetails(self):  
        #embed code into HTML using the f-string formatting method  
        output = f"""  
        <header>{self.getCustomerName()}</header>  
        <section class="mainSection">  
            <p><span class="label">Phone:</span> {self.getPhone()}</p>  
            <p><span class="label">Address:</span> {self.getAddress()}</p>  
        </section>  
        """  
  
        #return the results of the HTML() method  
        return HTML(output)
```

If you recall from previous chapters, the output of code cells is fairly bland. With the above code, the output is more dynamic and interesting. You now possess enough knowledge to stylize your notebooks to make them look professional. You can use stylized notebooks for presentations of analytical work, or to share with a team. For example, you can add `<h1>`, `<h2>`, `<header>`, or other heading tags to create a title for the cell output.

## Formatting Financial Data

To this point, we have simply printed financial data as a float data type. At times, you have seen examples where a dollar sign was added to a numeric value representing a dollar amount using f-strings. This method is fine for early analysis where presentation isn't important. However, as you begin to present your analyses to others, you want financial data to be formatted correctly. Financial data without commas in the correct place is harder to read, particularly for large publicly traded firms that account for billions of dollars in their financial statements. It is worth taking a moment to examine ways to format financial data to make it more readable.

Formatting financial data is simple with the `locale` library in Python. This library will choose the appropriate currency formatting for a particular locale. For example, in the US, commas are used to separate numbers in financial data with decimal points represented by periods. In other countries, it is the opposite. That is, periods separate numbers and decimal points are represented with commas. To use the `locale` library you must first import it and set the locale. The first line of code below imports the library. The second line of code sets the locale for all of the `locale` features to the locale set for your system. For the author, these settings return: 'en\_US.UTF-8' for US English.

```
import locale  
locale.setlocale(locale.LC_ALL, "")
```

Once the locale library is imported and the locale set, you can begin to format financial data with the currency() method. If you wish to include commas to separate numbers at the thousands, millions, billions, etc. you will also need to set the grouping parameter to True. For example:

```
portfolioValue = 12434521.34  
print(f"The portfolio earned {locale.currency(portfolioValue, grouping=True)} this year")
```

The code above would output: The portfolio earned \$12,434,521.34 this year. Although simple to use, formatting financial data can make reports far easier to read.

## Chapter 8 Assignment

Please see the [Chapter Assignments Workbook](#) for the Chapter assignment.

# Chapter 9:

## Creating Interactive Notebook Interfaces

Now that you possess an understanding of basic programming logic (e.g., variables, methods, objects, conditions, loops, etc.), interface development tools (e.g., HTML and CSS), and data analytics tools (e.g., pandas and matplotlib), you are ready to create more interactive notebooks.

Although many notebooks will simply be used to perform calculations on specific datasets, notebooks can include greater interactivity for repeated analyses. For example, instead of writing new code every time you want to plot the time series of price data for a particular stock, you could enter the stock ticker for a stock into a textbox and have the program lookup that stock and plot the data. If you are a financial analyst who wants to perform custom calculations on a number of stocks over time, you may need to write your own scripts. The same is true for many other business positions. Programs can make repeated calculations quick and painless. An interactive notebook can help you automate parts of your work while you wait for the IT department to find the time and resources to provide a more robust software tool.

This chapter will explore adding dynamic elements to your notebooks. Creating dynamic and interactive notebooks will make the notebooks themselves more useful to a greater number of users, including to yourself. To begin, we will learn about event driven programming. Then, we will explore HTML tags that will help you create interactivity in an interface. Last, we will examine how these interactive HTML elements can be used within notebooks to integrate user input into analytics notebooks.

### Event Driven Programming in Notebooks

To this point, you have learned about HTML container tags (e.g., div, header, footer, etc.) and other content tags (e.g., p, img, buttons, etc.). With this information, you should be able to create a simple user interface. However, despite showing buttons and other components, the interfaces you can create aren't interactive. To allow users to interact with page elements and to perform actions, you must code events into the program.

Many types of events can exist within a computerized system, namely: external events, temporal events, and state events. **External events** are events triggered by a human actor (e.g., clicking a button, pressing a keyboard key, etc.) or by another external information system. **Temporal events** are events triggered by temporal triggers. For example, if you create a calendar event in a Google calendar, it might send you a reminder 30 minutes before the event. Or you may wish for a system to automatically generate and email you a quarterly report at the end of a quarter. **State events** are similar to temporal events except that state events are triggered by changes in the state of an object, usually in relation to some threshold value, rather than based on time. For example, assume you wish your system to automatically order more inventory if inventory levels drop below a specified threshold. The system could check inventory levels, compare current levels to the threshold value, and then submit an automated order when inventory levels fall below the threshold. State events increasingly rely on analytics and machine learning to identify thresholds and to make decisions about when to perform specific behaviors.

Temporal and state events can be embedded into a system, but have little to do with the user interface. The user interface has more to do with external events (i.e., user input). This chapter focuses on external events triggered by human actors interacting with user interface components in notebooks. This chapter will explore interface elements and libraries that will allow you to track mouse click events, and other user interactions. In this chapter, we will explore a few of the many interactive interface components and events that can exist within HTML pages and notebooks. Once you understand the fundamentals of interactive interface components and their associated events, you will be able to learn more about a variety of other components and events on your own.

**Event driven programming** is concerned with responding to external events that occur within your program. When users click their mouse over a button, press a keyboard key, or interact with a user interface element in some other way, an event listener makes note of the click, keypress, or other action. The event listener calls the appropriate event handler to perform a specific behavior. An **event listener** is a method within a program that waits for user input, such as a mouse click or key press. The event listener is associated with an event handler. An **event handler** is a method that contains the logic that should be performed when a specific event is triggered.

For example, an event listener could be created by the developer to notice a mouse click on a "submit button" within an online form. The event listener would call the event handler that belongs to the submit button. The event handler would run its code, such as saving the information in the form to a database. This chapter will show you how to create interactive components and handle user initiated events in notebooks.

## Interactive HTML Elements

HTML possesses a series of tags that support interface interactivity. In the previous interface chapter, you learned how to include buttons in an interface, which are one type of interactive element. You also learned about some CSS, such as pseudo-classes, to give the interface the appearance of interactivity. However, you were not introduced to many interactive HTML elements, nor how to create true interactivity. This section will explore additional HTML elements that support interactivity. Later sections in this chapter will explain how to use these elements within notebooks. However, learning about interactive HTML elements will help you create traditional systems should you choose to move away from notebooks.

### **Form field elements for user interaction**

Most business systems contain **forms** that allow users to enter and submit information, such as username/password forms, account registration forms, profile creation forms, shopping cart checkout forms, etc. Forms consist of **form fields**, such as text boxes, sliders, dropdown menus, and buttons, that allow users to enter information and interact with the interface.

Many HTML form field elements contain a **name property**. The value of the name property is sent to the backend programming code along with the value of the form field. This is how the backend is able to process data from the frontend interface. For example, a textbox form field might have a name of "firstName". Suppose a user enters "Taylor" into the textbox and clicks a button to submit the form. The user interface would send a message to a backend programming script that would associate firstName with Taylor: firstName=Taylor. The backend could then add Taylor to a database or process the

information in other ways. Many form elements also contain an **id property** so that each form element can be uniquely identified from other forms fields. The ids can also help you manage the interactivity of forms.

Below you will find a variety of form fields that you can add to an interface to increase user interaction and collect necessary business data.

### Some Common HTML Form Field Tags

**The `<form>` tag.** Used to contain all of the form fields for a specific form. Form field elements go between the opening and closing form tags. Form tags have two primary properties: action and method. The action property designates a backend programming script to call that will process the form data on the backend server when the form is submitted. The method property designates how the data will be sent to the backend server. For our purposes, we will primarily use the “post” data transmission method. Usage:

```
<form action="DataProcessing.py" method="post"> </form>
```

**The `<fieldset>`, `<legend>` tag.** Used to break large forms into separate and related sections of form elements. Fieldsets wrap sections of a form with a border and title. The title is created with the legend tag, which is embedded inside the fieldset tag. For example, one section of a form might contain elements to enter shipping information. Another section might contain elements related to payment information. Fieldsets can visually differentiate these areas of a form for users. Elements of the form go within the fieldset tags. Usage:

```
<fieldset>
    <legend>Title of the Fieldset</legend>
</fieldset>
```

**The `<input>` tag.** Used to create a number of different form elements. The input tag can be changed to different types of form fields through the “type” property. Common input field types include: text, radio, checkbox, submit, and button. Each of these types is highlighted in this section. Input tags do not require closing tags. Input tags belong within the form tags and may also be embedded within fieldset tags if fieldsets are used. Usage:

```
<form action="DataProcessing.py" method="post">
    <fieldset>
        <legend>Profile Information</legend>
        <input type="text" id="firstName" name="firstName"><br>
        <input type="text" id="lastName" name="lastName">
    </fieldset>
</form>
```

**The `<label>` tag.** Used to make forms usable for individuals with visual impairments. To create an accessible interface, a label tag **should exist for every input tag**. In doing so, screen-reader software can help individuals with visual impairment fill out the form. A label connects to a specific form field by matching the id of the form field with the “for” property of the label. Usage:

```
<label for="firstName">First Name:</label> <input type="text" id="firstName" name="firstName">
```

**The `<input type="text">` tag.** Used to create single-line text fields where users can enter information via their keyboards. You should include an id property and a name property. For accessibility purposes, include an associated label as well. Usage:

```
<label for="lastName">Last Name:</label> <input type="text" id="lastName" name="lastName">
```

**The `<input type="tel">` tag.** Used to create single-line text fields where users can enter phone numbers via their keyboards. You can enter a pattern for the phone numbers that some browsers will validate before submitting the form. You should include an id property and a name property. For accessibility purposes, include an associated label as well. Usage:

```
<label for="phone">Cell Phone:</label> <input type="tel" id="phone" name="phone" pattern="^\([0-9]{3}\)[0-9]{3}-[0-9]{4}">
```

**The `<input type="email">` tag.** Used to create single-line text fields where users can enter email addresses via their keyboards. Some browsers will validate that a proper email format was entered before submitting the form. You should include an id property and a name property. For accessibility purposes, include an associated label as well. Usage:

```
<label for="email">Email:</label> <input type="email" id="email" name="email">
```

**The `<input type="password">` tag.** Used to create single-line text fields where users can enter passwords via their keyboards. The password text is not shown as the user types. Instead, asterisks or round bullets replace text characters. You should include an id property and a name property. For accessibility purposes, include an associated label as well. Usage:

```
<label for="password">Password:</label> <input type="password" id="password" name="password">
```

**The `<input type="date">`, `<input type="time">`, `<input type="datetime-local">` tag.** Used to create single-line text fields where users can click to open a calendar, time-picker, or calendar and time-picker. You should include an id property and a name property. For accessibility purposes, include an associated label as well. Usage:

```
<label for="startDate">Date:</label> <input type="date" id="startDate" name="startDate">  
<label for="reminder">Reminder:</label> <input type="time" id="reminder" name="reminder">  
<label for="appt">Appointment:</label> <input type="datetime-local" id="appt" name="appt">
```

**The `<input type="file">` tag.** Used to create single-line text fields with a browse button where users can search for files on their computers to upload files to a system. You should include an id property and a name property. For accessibility purposes, include an associated label as well. Usage:

```
<label for="photo">Profile Picture:</label> <input type="file" id="photo" name="photo">
```

**The `<input type="color">` tag.** Used to open a color-picker where users can select a color. You should include an id property and a name property. For accessibility purposes, include an associated label as well. Usage:

```
<label for="color">Color:</label> <input type="color" id="color" name="color">
```

**The `<input type="radio">` tag.** Used to create grouped radio buttons that permit only a single alternative to be selected from a group of choices. You should include an id property and a name property. The name property should be the same for every radio button in the group. For accessibility purposes, include an associated label for each radio button in the group. Usage:

```
<p>Include a 2-year warranty with your purchase?</p>
<input type="radio" id="yesWarranty" name="warranty"> <label for="yesWarranty">Yes</label><br>
<input type="radio" id="noWarranty" name="warranty"> <label for="noWarranty">No</label>
```

The **<input type="checkbox"> tag**. Used to create an option that can be selected or deselected. You should include an id property and a name property. For accessibility purposes, include an associated label for each checkbox. Usage:

```
<p>Check all features you would like included in your new vehicle?</p>
<input type="checkbox" id="heatedSeat" name="heatedSeat"> <label for="heatedSeat">Heated Seats</label><br>
<input type="checkbox" id="XMRadio" name="XMRadio"> <label for="XMRadio">XM Radio</label><br>
<input type="checkbox" id="parkingAssist" name="parkingAssist"> <label for="parkingAssist">Parking Assist</label><br>
```

The **<input type="submit"> tag**. Used to create a form submission button. You should include a value property, which sets the text that appears on the button. Usage:

```
<input type="submit" value="Create Account">
```

The **<select>, <option> tag**. Used to create a dropdown that users can click on to view a list of options. You should include an id property and a name property in the select tag. The options within the select dropdown are each contained within an option tag. Each option tag has a value property that is passed to the backend when the option is selected and the form is submitted. For accessibility purposes, include a label for the select tag. Usage:

```
<label for="warranty">Warranty Type:</label><br>
<select id="warranty" name="warranty">
    <option value="1yr">1 year warranty</option>
    <option value="3yr">3 year warranty</option>
    <option value="5yr">5 year warranty</option>
</select>
```

The **<textarea> tag**. Used to create a multi-line text area where users can write paragraphs of text. You should include an id property and a name property. The size of the textarea can be adjusted by setting the cols and rows properties. For accessibility purposes, include a label for the select tag.

Usage:

```
<label for="post">Write a Post:</label><br>
<textarea id="post" name="post" cols="50" name="15"></textarea>
```

With these commonly used form elements, you can create a variety of forms to improve the interactivity of an interface. The example below shows how these elements could be used to create a business form to collect supply chain partner information.

### HTML Form Elements to Create a Simple Form

```
<html>
<head>
```

```

<title>New Supply Chain Partner Form</title>
</head>
<body>
    <form action="process.py" method="post">
        <fieldset>
            <legend>Company Information</legend>
            <label for="companyName">Company Name:</label><br>
            <input type="text" id="companyName" name="companyName"><br>
            <label for="type">Company Type:</label><br>
            <select id="type" name="type">
                <option value="for-profit">For Profit</option>
                <option value="non-profit">Non-Profit</option>
                <option value="ngo">Non-Governmental Organization</option>
            </select><br>
            <label for="details">Company Details:</label><br>
            <textarea id="details" name="details" cols="50" name="15"></textarea>
        </fieldset>
        <fieldset>
            <legend>Company Rep Information</legend>
            <label for="firstName">First Name:</label><br>
            <input type="text" id="firstName" name="firstName"><br>
            <label for="lastName">Last Name:</label><br>
            <input type="text" id="lastName" name="lastName"><br>
            <label for="phone">Cell Phone:</label><br>
            <input type="tel" id="phone" name="phone" pattern="\\([0-9]{3}\\)[0-9]{3}-[0-9]{4}"><br>
            <label for="email">Email:</label><br>
            <input type="email" id="email" name="email">
        </fieldset>
        <input type="submit" value="Submit">
    </form>
</body>
</html>

```

If the form in the example above were opened in a browser, a user could enter information into the form and click the submit button. Upon clicking the submit button, the information from the form would be sent to the process.py script specified in the opening <form> tag. The process.py tag would exist on a backend server and contain logic to store the information in a database. This simple example provides a glimpse into the way programs handle user input. With your basic understanding of HTML forms and form elements, we will now explore how interactive form elements can be embedded into a notebook. For simplicity, we will utilize a Python library, **ipywidgets**, that will help to embed interactive form elements into notebooks.

## Interactive Form Elements in Notebooks

Notebooks possess an underlying kernel that manages interactivity within the notebook. This kernel makes it more difficult to create interactive forms in notebooks than in traditional websites. Recall that notebooks run in a browser as a web page. Thankfully, other developers have written libraries to embed interactive form elements into notebooks to simplify the use of forms that are compatible with the notebook kernel. In this section we will explore the ipywidgets library as a means to add interactive form elements to a notebook.

### ***Using ipywidgets for interactivity***

The complete documentation for the ipywidgets library can be found at the following URL:

<https://ipywidgets.readthedocs.io/en/stable/>.

By this point, you have probably read a number of forums and some documentation in order to complete the associated programming assignments. Although we will cover some examples in this section, we will not cover the installation and complete usage of ipywidgets. It is important that you learn to reference documentation to complete tasks. If ipywidgets is not included with your Anaconda installation, you should click the documentation link above to learn how to install the library. It can be installed with a simple pip command.

You will review many technical documents as a tech-savvy business professional. It is important that you practice now. Don't be discouraged if reading technical documentation doesn't come to you naturally. It takes time to learn how to read and process technical documentation. Although good documentation is as friendly to read as possible, that doesn't mean good documentation is simple. Be patient with yourself. View each piece of technical documentation you read as practice. With time, reading technical documents will become second-nature. After installing ipywidgets, simply import the library into a notebook:

```
import ipywidgets as widg
```

### ***List of useful interactive widgets***

The ipywidgets library consists of a variety of different widgets that are based on HTML form elements.

The table below presents some of the widgets that you may find useful for analytics projects. For a complete list of the widgets, visit:

<https://ipywidgets.readthedocs.io/en/stable/examples/Widget%20List.html>.

#### **Useful Widgets for Notebook Interactivity**

Notebooks used for business analytics will contain a variety of numbers (i.e., integers and floats). At times, you may wish the user (i.e., yourself or your peers) to be able to experiment with different numeric values. When conducting an analysis, you often want to know how sensitive your model is to changes in key assumptions. For example, if you are examining capital investment opportunities, you may want to alter the internal rate of return (i.e., an assumption in your analysis) when calculating the net present value of dollars in a financial analysis. The internal rate of return (IRR) isn't always a clear cut number. Some numbers may be overly conservative, while others may be overly

enthusiastic. Being able to experiment with the value to understand how the IRR affects the outcomes of a financial analysis can help you make better financial decisions. Many other values, like IRR, are included in a variety of business analyses. Be willing to experiment with these values to understand how they affect the outcomes. This experimentation with values that represent model assumptions is called **sensitivity analysis**. Numerically oriented widgets can allow you to change the values for key numeric assumptions with an easy to use interface element. The alternative would be changing the value manually in the code itself.

### Labels

**Label()** is a widget that places a textual label in the interface. The most useful property of the label is: **value** - the string value you want to display on the screen to the user. Labels are used to provide information to the user.

```
import ipywidgets as widg  
IRRLabel = widg.Label(value="Experiment with IRR values:")
```

### Numeric Sliders

Numeric sliders are a common interface element (made from a combination of HTML tags) that allow users to easily select a number from within a range of numbers. The ipywidgets library has two built-in classes for numeric sliders: **IntSlider()** and **FloatSlider()**.

**IntSlider()** is a widget that places a slider element in the interface that has a bounded range (i.e., min and max values) of integers that can be used to select an integer value from the range. The **IntSlider()** class has a number of attributes that you can set to change the look and behavior of the slider. Some useful properties include: 1) **value** - the default value within the range of values where the slider will start. Also used to extract the value the user selects, 2) **min** - the minimum value on the slider, 3) **max** - the maximum value on the slider, 4) **step** - the size of jump between numbers on the slider (e.g., change by 1, 5, 10, etc.), 5) **description** - text placed to the left of the slider to label it for clarity. Other properties exist as well, but these will be the most commonly changed properties.

**Example Usage:**

```
import ipywidgets as widg  
IRRIntSlider = widg.IntSlider(value=5, min=0, max=10, step=1, description="IRR Value:")
```

**FloatSlider()** is a widget that places a slider element in the interface that has a bounded range (i.e., min and max values) of floats that can be used to select a decimal value from the range. The **FloatSlider()** class has a similar set of properties as the **IntSlider()**, including 1) **value**, 2) **min**, 3) **max**, 4) **step** - the size of jump between numbers on the slider. Can be fractional steps (e.g., 0.01, 0.1, 0.5, 1.5, etc.), 5) **description**. Other properties exist as well, but these will be the most commonly changed properties. **Example Usage:**

```
import ipywidgets as widg  
IRRFloatSlider = widg.FloatSlider(value=5, min=0, max=10, step=0.5, description="IRR Value:")
```

### Numeric Text Boxes

Text boxes can also be used to allow users to input numeric values into your notebook. The numeric text box widgets ensure that users can only enter numbers. These text boxes will not allow users to

enter text. This protection prevents errors from occurring in your program (e.g., multiplying the number 6 by the word “hello”). The ipywidgets library has two built-in classes for numeric text boxes: IntText() and FloatText(). BoundedIntText() and BoundedFloatText() widgets also exist that allow you to limit the range of numbers accepted by a numeric text box. See the documentation for details on bounded text boxes.

**IntText()** is a widget that places a text box element in the interface that only accepts integer values. The IntText() class has a couple of important properties, including: 1) **value** - the default value to display in the text box when the page loads. Also used to extract the value the user enters, 2) **description** - text placed to the left of the text box to label it for clarity. Other properties exist as well, but these will be the most commonly changed properties. **Example Usage:**

```
import ipywidgets as widg  
IRRIntText = widg.IntText(value=5, description="IRR Value:")
```

**FloatText()** is a widget that places a text box element in the interface that only accepts decimal values. The FloatText() class has a couple of important properties similar to IntText(), including: 1) **value** and 2) **description**. **Example Usage:**

```
import ipywidgets as widg  
IRRFloatText = widg.FloatText(value=5.5, description="IRR Value:")
```

### Other Text Boxes

Beyond numeric text boxes, the ipywidgets library has a variety of other text box outputs derived from HTML elements. For example, a string text box can be used to collect textual data from users (e.g., to lookup data for a stock ticker). Similarly, a date picker text box can be used to open a calendar to specify a date range from which to select data for an analysis. Many text oriented widgets exist, we will explore the Text(), Textarea(), DatePicker(), and FileUpload() widgets.

**Text()** is a widget that allows users to enter a short string value to use in a notebook. Text widgets might be used to enter a term to search for within a data set. The Text() class has a few important properties, including: 1) **value** - the default value to display in the text box when the page loads. Also used to extract the string value the user enters, 2) **placeholder** - a placeholder value that doesn't act as a default value, but fills the text box to provide cues to the user, and 3) **description** - text placed to the left of the text box to label it for clarity. Other properties exist as well, but these will be the most commonly changed properties. **Example Usage:**

```
import ipywidgets as widg  
TickerText = widg.Text(placeholder="Enter a Ticker Symbol", description="Ticker Symbol:")
```

**Textarea()** is a widget that allows users to enter a longer string value to use in a notebook. Textarea widgets might be used to enter feedback or notes into a form. The Textarea() class has a few important properties similar to the Text() widget, including: 1) **value**, 2) **placeholder**, and 3) **description**. **Example Usage:**

```
import ipywidgets as widg  
NoteText = widg.Textarea(placeholder="Enter Notes About Analysis", description="Notes:")
```

**DatePicker()** is a widget that allows users to enter a date value to use in a notebook through either a text box or calendar. Two DatePicker widgets might be used to enter a date range to select from a subset of data in a dataset. The DatePicker() class has a couple of important properties, including: 1) **value** - used to extract the date value the user selects (the date will be in datetime.date() format) and 2) **description** - text placed to the left of the text box to label it for clarity. Other properties exist as well, but these will be the most commonly changed properties. **Example Usage:**

```
import ipywidgets as widg  
StartDate = widg.DatePicker(description="Start Date:")
```

**FileUpload()** is a widget that allows users to browse to and upload files from their file system into a notebook. The FileUpload() class has a few important properties, including: 1) **value** - used to extract information about the file(s) that were uploaded, 2) **data** - used to extract just the file contents of the files. It is returned as a list of contents for each file uploaded, 3) **accept** - sets the file extensions that can be selected (e.g., ".xlsx", ".xls", ".csv", etc.), 4) **multiple** - a True or False value that determines whether one or multiple files can be uploaded.. Other properties exist as well, but these will be the most commonly changed properties. **Example Usage:**

```
import ipywidgets as widg  
Upload = widg.FileUpload(accept=".xlsx", multiple=False)
```

### Boolean and Selection Widgets

In the chapter about flow control, you learned about how to use if() statements to manage options and alternatives within a program. Boolean and selection widgets allow users to choose from options and alternatives through a user interface. If() statements can then be used to execute logic based on the selected option or alternative. Many boolean and selection widgets exist, but we will explore the Checkbox() boolean widget, and the Dropdown() selection widget.

**Checkbox()** is a widget that allows users to choose an option or not (i.e., a True/False boolean decision). For example, when conducting a financial analysis, you might want to let the user choose between using raw dollar amounts or NPV adjusted dollar amounts. The Checkbox() class has a couple of important properties, including: 1) **value** - the default value to display when the page loads (i.e., True is checked, False is unchecked). Also used to extract the boolean value based on the user selection and 2) **description** - text placed to the right of the checkbox to label it for clarity. Other properties exist as well, but these will be the most commonly changed properties. **Example Usage:**

```
import ipywidgets as widg  
NPVChecker = widg.Checkbox(value=True, description="Conduct NPV Analysis?")
```

**Dropdown()** is a widget that allows users to choose an alternative from a dropdown box filled with available alternatives. For example, your program could be written to perform different types of statistical analyses. The dropdown could be used to perform the selected analysis. The Dropdown() class has a couple of important properties, including: 1) **options** - a Python list with the alternatives to include in the dropdown box. 2) **value** - the default value to display in the dropdown when the page loads. Also used to extract the selected value when the user makes a choice, and 3) **description** -

text placed to the left of the dropdown to label it for clarity. Other properties exist as well, but these will be the most commonly changed properties. **Example Usage:**

```
import ipywidgets as widg
AnalysisType = widg.Dropdown(options=["Regression", "Decision Tree", "SVM"], value="Regression",
description="Select an Analysis: ")
```

### Action Widgets

Action widgets, such as buttons, allow a user to perform an action that can execute a broader set of logic. An action widget allows users to perform complex logic, such as making automated analysis decisions based on multiple other widgets. For example, you could write code to cause a button click to extract the ticker symbol from a Text() widget and the date from a DatePicker() to collect data for a particular stock. We will examine the Button() widget below.

**Button()** is a widget that allows users to execute an action. The Button() class has a couple of important properties, including: 1) **description** - text placed within the button to label it for clarity and 2) **tooltip** - text that pops up when the user hovers over the button to explain what action will be performed when clicked. Other properties exist as well, but these will be the most commonly changed properties. **Example Usage:**

```
import ipywidgets as widg
SaveButton = widg.Button(description="Save", tooltip="Click to Save the Results of the Analysis")
```

With the widgets described above, you now possess enough knowledge to build an interactive interface within your notebooks. We now turn to widget events to help you execute logic when an event is triggered by the user (i.e., a button click). To make each widget display in a notebook cell, you need to pass the widget to the display() method built into Jupyter notebooks. The example below shows the creation and display of multiple widgets. Widgets can also be mixed with other HTML code using the HTML library presented in an earlier chapter. You may need to wrap the HTML() method in a display() method as well to make it work with widgets. The widgets and HTML will display in the order that they are called in the display() methods.

### Displaying Widgets

In the simple example below, an HTML widget is mixed with Text(), FileUpload(), and Button() widgets to create an interactive interface.

In one of the first cells, the two libraries would need to be imported:

```
import ipywidgets as widg
```

In another cell, the HTML and widgets would be created and displayed:

```
HTMLCode = widg.HTML("<h1>Set Up a New Analysis</h1>")
AnalysisTitleText = widg.Text(placeholder="Analysis Title", description="Title:")
UploadedData = widg.FileUpload(accept=".xlsx", multiple=False)
SaveButton = widg.Button(description="Save", tooltip="Click to Save the Results of the Analysis")
```

```
display(HTMLCode, AnalysisTitleText, UploadedData, SaveButton)
```

The code would display the following to the output.

```
In [67]: HTMLCode = HTML(f"<h1>Set Up a New Analysis</h1>")
AnalysisTitleText = widg.Text(placeholder="Analysis Title", description="Title:")
UploadedData = widg.FileUpload(accept=".xlsx", multiple=False)
SaveButton = widg.Button(description="Save", tooltip="Click to Save the Results of the Analysis")
display(HTMLCode, AnalysisTitleText, UploadedData, SaveButton)
```

### Set Up a New Analysis

Title:

## Triggering events with on\_click() listeners

Event driven programming requires that you write the program to listen for events in the interface (e.g., a mouse click on a button) and handle the logic to perform when an event is triggered. The ipywidgets library makes these steps fairly simple. The Button() widget has a simple on\_click() method that can be called on a Button() widget object to trigger programming logic. For example:

```
import ipywidgets as widg
SaveButton = widg.Button(description="Save", tooltip="Click to Save the Results of the Analysis")
SaveButton.on_click(nameOfMethodToCall)
```

The programming logic that the on\_click() method triggers is contained within a separate Python function that you must create. In the example above, the name of this function you would need to create is nameOfMethodToCall(). Notice that the name of the method in the on\_click() method does not end with parentheses. You simply pass the name of the method as an argument to on\_click().

In the example below, you will see how to create an event handler function that is called by the on\_click() method when a click occurs. You will notice in the example the use of an **Output()** widget. The Output() widget allows you to display content from interactive events in a cell's output by updating the content displayed in the cell when the event is triggered. **Without the Output() widget, you cannot update the cell output when an event is triggered via a Button() widget click.**

The updating of the cell output happens with the “**with output:**” statement. In the code below, you will see the with output: statement in the stockDataEventHandler() method. The with output: statement should go in whichever method is used to change the user interface. If your event-triggered display() methods are not contained within the with output: statement, the interface will not change when a user clicks a button.

When using on\_click() events to change cell output, you may need to call the clear\_output() method built into Jupyter notebooks within the event handler method. Without the clear\_output() method, each time you click the button, the output will be duplicated instead of simply replaced. The clear\_output() method will get rid of the old output in place of the new output.

In the example, the plot() method is called on the pandas data frame to plot the stock time series data. This plot is created from the matplotlib library. Plots created via matplotlib do not display as other elements. To make the plots clear the output area when clear\_output() is called, you must import the following library:

```
from ipywidgets.widgets.interaction import show_inline_matplotlib_plots
```

This library makes plots display in the same fashion as other elements allowing them to be cleared from the output when clear\_output() is called.

### Triggering Events with on\_click() Events

In the simple example below, A Text() widget, two DatePicker() widgets, and a Button() widget are used to collect stock data from the Alpha Vantage API and display a time series plot. The Output() widget updates the cell output with the stock data.

In one of the first cells, the four libraries would need to be imported:

```
import ipywidgets as widg
from ipywidgets.widgets.interaction import show_inline_matplotlib_plots
from iPython.core.display import HTML
import pandas_datareader as pdr
```

In another cell, the HTML and widgets would be created and displayed:

```
#create Output object to change cell output
output = widg.Output()

#create other visual widgets and HTML code
heading = HTML(f"<h3>Enter a Stock Ticker to Retrieve Price Data</h3>")
ticker = widg.Text()
startDate = widg.DatePicker(description="Start")
endDate = widg.DatePicker(description="End")
button = widg.Button(description="Get Data", tooltip="Click to Get Stock Data")

#display the widgets and HTML. The Output widget must also be displayed to work correctly
display(heading, ticker, startDate, endDate, button, output)

#set up the on_click event listener and tell it what function to call as the event handler
button.on_click(stockDataEventHandler)

#set up the event handler function the on_click() listener will call to
def stockDataEventHandler(b):
    #tell the event handler what to do with the cell output
    with output:
        clear_output() #clear the cell output so it doesn't duplicate when you click the button again
```

```

#use the DataReader to get stock data based on the text field and date picker values
df = pdr.DataReader(ticker.value, start=startDate.value, end=endDate.value,
                     data_source="yahoo")

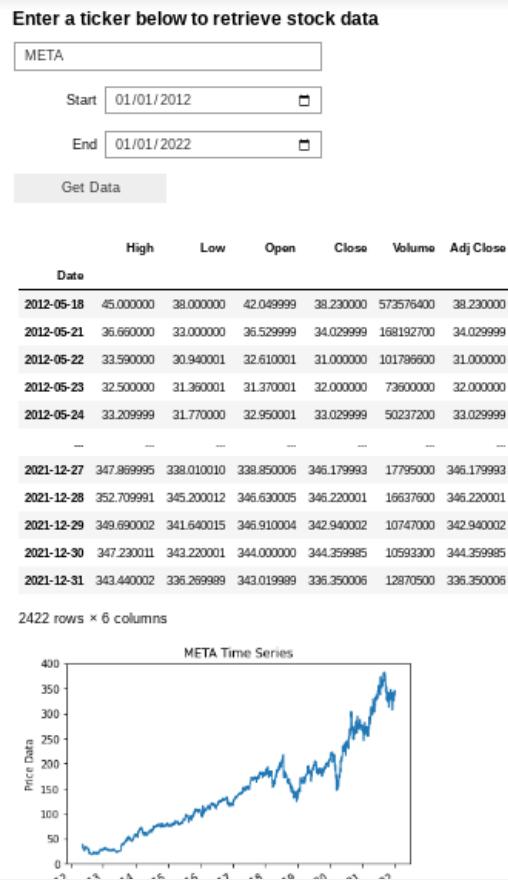
#display the stock data frame to the cell output
display(df)

#create the time series plot of the stock data frame
timeSeriesPlot = df["Adj Close"].plot()
timeSeriesPlot.set_xlabel("Date")
timeSeriesPlot.set_ylabel("Price Data")
timeSeriesPlot.set_title(f"{ticker.value} Time Series")

#display the plot in a way that it can be cleared by the clear_output() method
show_inline_matplotlib_plots()

```

With the code above, when the button is clicked, the data frame and time series plot for the requested stock ticker and date range will be displayed. By changing the value in the ticker Text() widget and the DatePicker() widgets, different stock data can be retrieved. The image below shows the output of the code when data is entered into the textboxes.



You now have an understanding about how to use the `on_click()` event listener to call an event handler method of your own creation. With this knowledge and creativity, you can develop interactive interfaces for a variety of business problems. For example, you can create calculators made from simple forms to enter information with programming logic to process the data from the forms.

## **Debugging with ipywidgets**

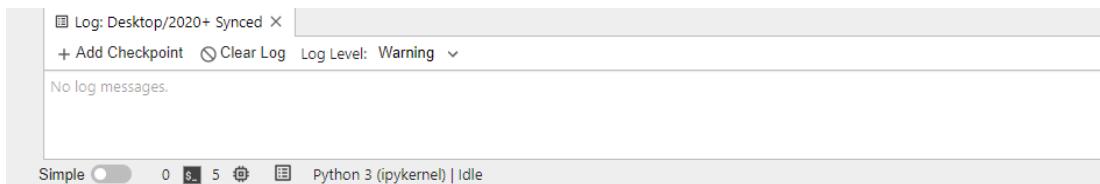
The `ipywidgets` library provides a graphical user interface. In doing so, it hides error messages. Many error messages no longer appear in the cell output when using the `ipywidgets` library. For this reason, you may need to open the error log in the Jupyter Lab interface to diagnose errors you encounter. Figure 9.1 shows the log icon highlighted. The icon looks a little like a book. It can be found in the bottom left-hand side of the Jupyter Lab interface.

**Figure 9.1: The Error Log Icon to Open the Error Log Window**



If you click the icon, a small window will open at the bottom of the screen. When you run your code, any hidden error messages will appear in the log area. The log stores error messages until they are cleared. If you aren't sure if you are looking at an old error message, click the Clear Log button in the error log window and run your cells again. Figure 9.2 shows the error log window.

**Figure 9.2: The Error Log Window**



If you don't see the error log icon at the bottom of Jupyter Notebook, you can access the error log from the "View" menu at the top of the Jupyter Lab interface. Simply click the Show Log Console option and the error log will appear.

Breakpoints will still work with the `ipywidgets` library, but the CallStack controls in the debug window will only be triggered when an event is triggered. If you have not properly set up your event listeners and event handlers, you may not be able to use the CallStack tools to move through the different breakpoints. In such cases, you may need to use the `print()` statement method of debugging described in Chapter 3. You will then use the error log window to monitor the `print()` statement output. Again, the `print()` statements will not print to the cell output when using widgets. The error log window will be useful to you as you begin working with user interfaces.

## **Utilizing data from event handlers**

The event handler method in the previous example named `stockDataEventHandler()` does not and cannot easily return a value. This is true for other event handlers you create to be called by the `on_click()` event listener. The `stockDataEventHandler()`, and other handler methods you create, are called by the `on_click()` event listener, which limits the ability to return a value to a variable as you are accustomed to when calling

a function. Further, if you try to change the value of a variable that exists outside of the handler within the handler, the resulting change will not take place. However, the results of an event handler method can produce results that you may wish to use in other parts of your notebook. For example, you might want to use the data frame in the previous example in other cells. So, if you cannot manipulate variables external to the event handler and you are unable to return a value from an event handler, how do you use information from the event handler in other parts of the notebook?

There are multiple ways to extract information from an event handler; we will explore two options.

### ***Using the Output widget to store and access data***

First, if you wish to program in a procedural manner as in the example above (i.e., without classes and objects), you can use attributes of the Output widget to store information created with the event handler function. Building on the previous example, you could store the data frame with the stock information in an attribute of the Output widget. Python is a flexible language that allows you to create new attributes after instantiation. Thus, you can add on any public attribute you would like to the Output widget. For example:

```
output = widg.Output()  
data = pd.read_excel("C:/Users/myuser/file.xlsx")  
output.df = data
```

The example above is only intended to show how an attribute can be added to an object to store data. A more complete example can be found below. In this example, the data is created in the event handler method. The Output object is then used to hold data created within the event handler method. The Output object can then be used in other cells to carry information from the event handler to other cells in the notebook.

#### **Using the Output Widget to Carry Information**

This example builds on the previous example. The import statements from the last example would need to be in one of the first cells. Some of the code from the previous example has been removed for brevity.

In one cell, the HTML and widgets would be created and displayed as before:

```
#create Output object to change cell output
```

```
output = widg.Output()
```

```
#the Text(), Datepicker(), and Button() widgets would be added and displayed here as before
```

```
#set up the on_click event listener and tell it what function to call as the event handler
```

```
button.on_click(stockDataEventHandler)
```

```
#set up the event handler function the on_click() listener will call to
```

```
def stockDataEventHandler(b):
```

```
    #tell the event handler what to do with the cell output
```

```
    with output:
```

```

clear_output() #clear the cell output so it doesn't duplicate when you click the button again

#use the DataReader to get stock data based on the text field and date picker values
df = pdr.DataReader(ticker.value, start=startDate.value, end=endDate.value,
                     data_source="yahoo")

#the data frame stored in df could then be stored in the Output widget
output.stockData = df

#the data frame and plot could then be displayed as before

```

In another cell, you could access the df data frame by calling to the stockData attribute of the Output widget, like this:

```
output.stockData.head()
```

Further analysis or data cleaning could be performed on the data frame from other cells using the stockData attribute to store the data. In this way, data can be carried from the event handler to other cells.

Using the method above, the data frame from the event handler could be used throughout the notebook. If the user were to select a different stock using the Text() widget and click the button again, the output.stockData attribute would be updated with a new data frame for the selected stock. By rerunning the other cells that call the output.stockData attribute, the results of those cells would update as well. Using the Output widget for data storage allows you to connect your interactive interface with the rest of your notebook cells.

**You can also use the Button widget to pass information.** The Button object is passed to the event handler message each time a button is clicked. If you add an attribute to your Button widget, you can access the attribute in the event handler method through the instance of the Button. The example below shows how you can use a Button widget to pass data into an event handler when the Button is clicked.

### Using a Button Widget to Carry Information

```

button = widgets.Button(description="Click")

#add a custom attribute to the button object
button.source = "yahoo"

button.on_click(stockDataEventHandler)

#the parameter b below is the instance of the button that was clicked
def stockDataEventHandler(b):
    with output:
        clear_output() #clear the cell output so it doesn't duplicate when you click the button again

```

```
#notice that b.source is called for the data_source parameter.  
#this would retrieve the value from the custom source attribute of the button  
df = pdr.DataReader(ticker.value, start=startDate.value, end=endDate.value,  
                     data_source=b.source)
```

## Using classes to store and access data

The second method to make event handler data available to other cells requires the use of classes, or at minimum one class. In this method, you can create View classes (i.e., user interface classes). In the simplest implementation, the View class can be used to store the interactive elements of the interface. The attributes of the View class can then be used to store and share data between the event handler and the rest of the notebook. Alternatively, you can use a more complex architecture to move the data from View classes to Model/Domain classes using a Controller/Presenter class. In an earlier chapter, systems architecture was mentioned briefly. The idea of Controller/Presenter, Model/Domain, and View classes was introduced. As a reminder, View classes contain logic pertaining to the user interface (e.g., form elements, buttons, event listeners and handlers, etc.); Model/Domain classes store data about business objects; and Controller/Presenter classes control the logic that connects different Domain classes together and allows Domain classes to interact with View classes.

The code below shows a simple passive MVP framework in which the View class interacts with a Controller class, which in turn interacts with a Domain class. In the particular example, the View class is named StockHistoryView. The StockHistoryView class sends commands to the StockController class. The StockController class then manages commands sent to the Stock class. Through this architecture, the on\_click() event triggers the View to call the Controller. Then the Controller passes the data to the Domain class to be searched and stored in an attribute for later use. The Domain object returns the data to the Controller and the Controller returns the data back to the View to be displayed. The data can then be accessed in later cells by again calling the View class to request the data from the Controller, which then retrieves the data from the Domain class and returns it to the View class.

Although this particular architecture is overkill for the simplicity of this example, it offers you a glimpse into how some systems can be designed. The architectures can be even more complex than what you see in the example below, as this is a simplified implementation of an architecture.

### Using Classes to Carry Information

This example builds on the previous example. The import statements from the last example would need to be in one of the first cells. For brevity, parts of the code are left out. Each class would consist of more attributes and methods.

```
#the View class to build the user interface  
class StockHistoryView:  
    #the Controller is instantiated within the View class  
    def __init__(self):  
        self.controller = StockController()
```

```

#create widgets and HTML and store as attributes
self.heading = HTML("<h3>Enter a ticker below to retrieve stock data</h3>")
self.ticker = widg.Text()
self.startDate = widg.DatePicker(description="Start")
self.endDate = widg.DatePicker(description="End")
self.button = widg.Button(description="Search", tooltip="Click to Get Stock Data")
self.button.on_click(self.viewStockData)
self.output = widg.Output()

def show(self):
    display(self.heading, self.ticker, self.startDate, self.endDate, self.button, self.output)

def clear(self):
    clear_output()

def viewStockData(self, b):
    with self.output:
        self.clear()

        #use the controller to retrieve the stock history
        data = self.controller.searchStockHistory(self.ticker.value, self.startDate.value,
                                                    self.endDate.value)

        #display the data frame returned by the controller
        display(data)

        timeSeriesPlot = data["Adj Close"].plot()
        timeSeriesPlot.set_xlabel("Date")
        timeSeriesPlot.set_ylabel("Price Data")
        timeSeriesPlot.set_title(f"Time Series for {self.ticker.value}")
        show_inline_matplotlib_plots()

#the Controller class
class StockController:
    def __init__(self):
        self.stocks = {}
        self.currentTicker = None
    #other attributes would go here

#the controller class creates a Stock Domain object to retrieve and store the stock history
def searchStockHistory(self, ticker, startDate, endDate):
    stock = Stock(ticker)

    if ticker not in self.stocks:
        self.stocks[ticker] = stock

    self.currentTicker = stock

```

```

        return self.stocks[ticker].searchHistory(ticker, startDate, endDate)

    def getHistory(self):
        return self.stocks[self.currentTicker].getHistory()

#other methods would go here

#the Model/Domain class
class Stock:
    def __init__(self, tickerP):
        self.ticker = tickerP
        self.history = None
    #other attributes would go here

#use Alpha Vantage API to retrieve the requested history data
def searchHistory(self, ticker, startDate, endDate):
    av = AlphaVantageAPI("AlphaVantageAPIKeyGoesHere")
    av.getDaily(ticker)
    self.history = av.toDataFrame()
    #return only the data within the specified date range
    self.history = self.history[(self.history.index >= startDate) & (self.history.index <= endDate)]

    return self.history

#get the history after the history has been retrieved with the searchHistory method
def getHistory(self):
    return self.history

#other methods would go here

```

The code would be executed by instantiating the View class and calling the show() method. The code does the rest as the user interacts with the interface.

```

view = StockHistoryView()
view.show()

```

In another cell, you could access the stock history data frame created when a user searches for a stock by calling to the getHistory() method on the View object:

```

view.getHistory().head()
view.getHistory().plot()

```

Further analysis or data cleaning could be performed on the data frame from other cells using the getHistory() method, which returns the data frame. In this way, data can be carried from the event handler to other cells.

Although the example above is more complex than the simpler procedural example, it also provides you with a better understanding of systems architecture. Complex systems need some form of architecture to ensure that the many features in the system can be controlled and managed effectively.

With the knowledge you have gained from this chapter, you should now be able to integrate interactive elements into the user interface of a notebook. You should also be able to create event listeners and handlers. Last, you know how to utilize data created by the event handler to use throughout the notebook using both simple procedural and slightly more complex object-oriented architectures.

## Interactive Form Elements and Databases

Often, forms are used for storing and retrieving data from databases. You can use a notebook to help you store and retrieve database information using a graphical user interface. In this section, you will see another example of the MVP architecture presented in the previous example that includes a database. In this case, we will use an example of keeping track of customer information. A View class can contain multiple methods to create different interface pages. In the example below, the View class is used to display interfaces to: create a customer, list customers, and view one customer.

The database connections are managed with a database driver object as explained in Chapter 7. The database driver object is passed from the View to the Controller to the Domain class that uses it to store and retrieve data. This adds one additional piece to your information about possible architectures for systems.

### Using Interactive Interface Elements with Databases

This example builds a simple customer relationship management system. A real CRM system would be far more complex with many more classes and methods.

```
#the View class to build the several user interface pages
class CustomerView:
    def __init__(self, databaseDriverP):
        self.controller = CustomerController(databaseDriverP)

    #the interface for creating new customers
    def displayCreationInterface(self):
        #create widgets and HTML and store as attributes
        creationHeading = HTML("<h3>Create a New Customer</h3>")
        self.name = widg.Text(description="Name:")
        self.phone = widg.Text(description="Phone:")
        self.createButton = widg.Button(description="Create", tooltip="Click to Create the Customer")
        self.button.on_click(self.createCustomerEvent)
        self.createOutput = widg.Output()
        display(creationHeading, self.name, self.birthDate, self.createButton, self.output)

    #the event handler for the customer creation interface
    def createCustomerEvent(self, b):
        with self.createOutput:
```

```

clear_output()

#use the controller to create the customer
#check if customer was created successfully
if self.controller.createCustomer(self.name.value, self.phone.value,
    self.birthDate.value):
    display(f"The customer {customer.name} was created successfully!")
else:
    display("The customer could not be created at this time.")

#the interface for listing customers
def displayListInterface(self):
    #create and display interface information
    listHeading = HTML("<h3>Customers</h3>")
    display(listHeading)

    customers = self.controller.listCustomers()

    for customer in customers:
        html = HTML("""
            <div style="background-color: black; color: white;">{customer.name}</div>
            <div>
                <p>Birth Date: {customer.birthDate}</p>
                <p>Phone: {customer.phone}</p>
                <p>Possible Delete Button Here</p>
            </div>
        """)
        display(html)

#the Controller class
class CustomerController:
    def __init__(self, databaseDriverP):
        self.databaseDriver = databaseDriverP
        self.databaseDriver.connect()
    #other attributes would go here

#the controller class creates a Customer Domain object store in the database
def createCustomer(self, name, phone, birthDate):
    customer = Customer(self.databaseDriver)

    if customer.createCustomer(name, phone, birthDate):
        return True
    else:
        return False

def listCustomers(self):
    customer = Customer(self.databaseDriver)

```

```

return customer.listCustomers()

#other methods would go here

#the Model/Domain class
class Customer:
    def __init__(self, databaseDriverP):
        self.customerId = None
        self.name = None
        self.phone = None
        self.birthDate = None
        self.databaseDriver = databaseDriverP
    #other attributes would go here

#use the databaseDriver create() method to perform the SQL statement
def createCustomer(self, nameP, phoneP, birthDateP):
    statementString = "INSERT INTO Customer SET name=?, phone=?, birthDate=?"
    data = (nameP, phoneP, birthDateP)

    self.databaseDriver.createStatement(prepared=True)
    self.databaseDriver.executeStatement(statementString, data)
    self.databaseDriver.commit()

    if self.databaseDriver.getStatementRowCount() > 0:
        self.customerId = self.databaseDriver.getStatementInsertId()
        self.name = nameP
        self.phone = phoneP
        self.birthDate = birthDateP

def listCustomers(self):
    statementString = "SELECT * FROM Customer"
    self.databaseDriver.createStatement(dictionary=True)
    self.databaseDriver.executeStatement(statementString)
    result = self.databaseDriver.getStatementResult()
    return result

#other methods would go here

#The database driver interface and implementation would go here
#For brevity, these are not included. See Chapter 7 for examples.

Each page of the CustomerView interfaces could be created in different cells. For example, one cell
could contain the customer creation view like this:

view = CustomerView()
view.displayCreationInterface()

```

The customer list view could be displayed in the next cell. For example:  
`view.displayListInterface()`

Although long, this example shows how View, Controller, Domain, and DatabaseDriver classes can work together to provide interactive interfaces that interact with databases. Through such interfaces, you can add data to databases with simple forms, view data in databases, and more. This chapter has provided you with information about how to integrate the information you have learned in other chapters into notebooks in a way that non-technical users can use your code. You have also seen architectures that will help you build more traditional information systems. In the next section of the text, you will learn advanced topics to further help you integrate the knowledge you have gained to create more interesting and useful notebooks.

## Creating Containers for Interactive Widgets

The ipywidgets library does not interface well with the HTML libraries described in the previous chapter. Although it is possible to embed a widget into an HTML structure, it is more complicated than it needs to be. Instead of creating custom HTML structures to contain your interactive widgets, the ipywidgets library provides various container objects that you can use to contain interactive widgets. In the example in the previous section, the CustomerView had several interfaces that could be called in different cells. Alternatively, all of the interfaces could be contained in one cell with the use of container widgets.

Two simple, yet versatile container widgets are the **VBox** and **HBox** containers. The VBox container allows multiple widgets to be stacked on top of one another in a column. The HBox container allows multiple widgets to be placed side by side in a row. You can create more complex containers by embedding one container within another container. For example, two or more VBox containers could be contained within a HBox container (i.e., stacked side by side) to create a row with the VBox columns of widgets. You could also store multiple HBox containers in a VBox to obtain a similar effect.

The **Accordion container** allows you to store multiple widgets or container widgets within an expandable area. You can click on an Accordion container to show the widgets within it. You can click it again and the widgets will be hidden from view.

The **Tab container** is another useful container that behaves similar to the tabs in a web browser or in an Excel spreadsheet. Tab containers allow you to create multiple “pages” with different content. As such, Tab containers can be used to store multiple different interfaces within a small area. Figure 9.1 shows a Tab container with an Accordion container within it. As shown in the interface, other widgets are contained within the container widgets. In the figure, an HTML() widget for the black header, three Text widgets, and a Button widget are stored in a VBox container widget, which is stored in the Accordion container widget, which is stored in the Tab container widget. Through embedding of containers and widgets, complex interfaces can be created.

The VBox and Accordion containers share similarities in instantiation. In both cases, you pass a Python list of the widgets contained within the containers into the instantiation. For example:

```
vbox1 = widgets.VBox([widget1, widget2, widget3])
vbox2 = widgets.VBox([widget4, widget5, widget6])
```

```
accordion = widgets.Accordion([vbox1, vbox2])
```

The Tab container is instantiated and then the widgets stored within the tab are stored in an attribute named children. Each widget passed into the children attribute will represent a different tab in the interface. For example:

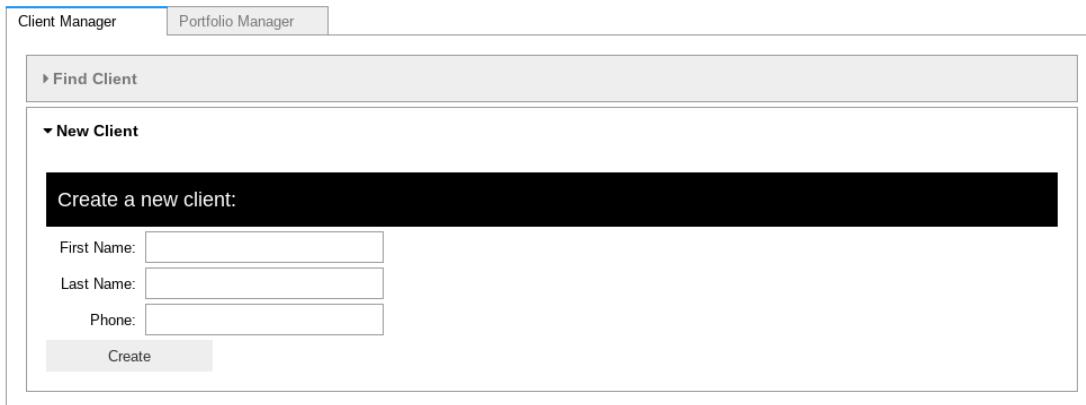
```
tabs = widget.Tab()  
tabs.children = [vbox1, vbox2]
```

The VBox, HBox, Accordion, and Tab container widgets all possess the children attribute. The next section will describe how the children widgets within a container widget can be updated after they are initially set.

The Accordion and Tab interfaces both possess titles for the accordions and tabs. The titles can be set for each accordion/tab using the set\_title() method. The set\_title() method accepts an index that represents the widgets passed into the Accordion or Tab widget in the order they were passed in, followed by the title to display on the accordion or tab. For example:

```
tabs = widget.Tab()  
tabs.children = [vbox1, vbox2]  
tabs.set_title(0, "Title for VBox1") #index 0 represents vbox1 above  
tabs.set_title(1, "Title for VBox2") #index 1 represents vbox2 above
```

**Figure 9.1: Tab Container Widget with an Embedded Accordion Container Widget**



## ***Changing the children widgets within a container***

At times, you will want to alter what widgets can be seen within a container widget after you have initially set the children of the container. These changes can create a more dynamic feel to the interface. These changes are usually triggered by a user clicking a button.

When initially setting the children widgets of a container, it appears they are stored in an array. In the previous section, you saw the following examples:

```
vbox1 = widgets.VBox([widget1, widget2, widget3])
```

or

```
tabs = widget.Tab()  
tabs.children = [vbox1, vbox2]
```

Although the initial children are set using an array, the ipywidgets library converts the array to a tuple. If you recall from earlier chapters, tuples are like arrays except that they are immutable data structures. That means that you cannot change the values inside of tuples or add values to tuples. So how do you change the children of a container widget if the data structure is an immutable tuple?

To change the children of a container widget, you must replace the existing tuple with another tuple. Assume that the original children were set up as follows:

```
tabs = widget.Tab()  
tabs.children = [vbox1, vbox2]
```

To change the children, such as adding another VBox, you would need to do the following:

```
tabs.children = (vbox1, vbox2, vbox3)
```

Notice that the data structure this time is a tuple with parentheses instead of an array with hard brackets. If you are simply adding to an existing set of widgets, you can also use the plus concatenation operator to create a new tuple that consists of the old tuple and an additional tuple. For example:

```
tabs.children = tabs.children + (vbox3, )
```

Recall that when a tuple only has one value, it needs a floating comma after the first item. The comm after vbox3 above is not a mistake.

Sometimes, you will want to reset the interface to its original configuration after adding an element to the children attribute. This can be done in a similar manner. You would simply set the children attribute equal to a tuple with the original widgets as items. Building on the previous example:

```
tabs.children = (vbox1, vbox2)
```

With the information in this chapter, you possess the knowledge to build simple interfaces and separate your code into different types of logic (i.e., view logic, controller logic, and model/database logic). With practice, you will possess the skills to create simple interfaces that you and your colleagues can use for tedious information tasks you repeat regularly in the workplace.

## Chapter 9 Assignment

Please see the [Chapter Assignments Workbook](#) for the Chapter assignment.

## Section IV: Advanced Programming Topics

This section of the text is dedicated to introducing some advanced programming topics. The purpose of these chapters is not for you to develop expertise in advanced programming techniques, but to provide you with a foundation to build upon. Section IV includes Chapters 10, 11, and 12.

**Chapter 10** will introduce you to more advanced forms of data collection and preprocessing. In Section II, you learned how to clean data and use the Alpha Vantage API to collect historical stock data. In this chapter, you will learn to access financial data in more flexible ways through web scraping and utilization of APIs. You will also learn some more advanced data cleaning tools and practices, such as converting nominal and ordinal data to numeric form and preprocessing textual data.

**Chapter 11** will introduce you to simple machine learning models. Section I introduced you to ideas about artificial intelligence and machine learning. This chapter will build on information shared in Section I by showing you how to build and train unsupervised and supervised machine learning models. You will learn about principal component analysis for dimensionality reduction. You will read about unsupervised cluster analysis models with a demonstration of how to use Python libraries to perform a k-means cluster analysis. You will also learn about several supervised learning models, including decision trees, random forests, and support vector machines with Python examples.

**Chapter 12** will introduce you to how businesses manage quality and efficiency in software development. In particular, this chapter introduces tools (i.e., git and GitHub) to assist with the continuous integration of code. Developing complex information systems requires multiple teams of programmers. This chapter explains how IT departments manage the code from multiple teams to ensure high-quality systems. You will learn to use the command line within your operating system to use git and GitHub, tools to assist with continuous integration efforts.

After completing this section, you will possess a greater understanding of the complexities of programming. You will not necessarily possess the expertise to be an advanced programmer, but you will better respect the knowledge and skill sets required of IT departments to develop business information systems. You will also develop further analytics capabilities, namely machine learning, that will assist you in making better business decisions with the data you will possess as a business professional. Continue to use the skills you have learned to prepare yourself to work in increasingly technical business environments. You can become a leader in business automation and analytics. It will require continued practice and effort.

# Chapter 10:

## Advanced Data Collection and Preprocessing

This chapter will introduce you to advanced ideas in data collection and data preparation (i.e., data preprocessing), such as web scraping and text preprocessing. You learned a little about data collection and preparation in Chapter 6. We will build on that foundation in this chapter. Being able to collect textual data is increasingly important. Unstructured textual data contains important information that is often ignored in business decisions. Learning to collect this data can help you conduct more interesting analyses. Similarly, learning how to preprocess textual data is important. This chapter will build on the data cleaning tools provided in earlier chapters. You will also learn how to collect financial information from application programming interfaces (APIs).

### Advanced Data Collection

Not all data is neatly contained within SQL databases owned by the organization using the data. SQL databases are highly structured with a simple query language. However, these structured databases represent only a small amount of all data available for analysis. For example, the Internet is littered with textual, image, and video data. Although this data is available for analysis, it is not well structured. Extracting this data is not as easy as writing a SQL statement to collect data from a database. Instead, you must “scrape” the data from the web to build repositories of unstructured data. Possessing web scraping skills will help you bring different forms of data into an analysis. Doing so will help you make better data driven decisions.

Further, data can be purchased from other organizations that collect data. Your company will likely purchase data from many other organizations. Importantly, you don't often purchase data that comes in an Excel file. Instead, an organization that collects information makes that information available for purchase to other companies through an application programming interface (API). An API is simply a language for requesting information that a computer can read. You are used to using graphical user interfaces to retrieve information. An API is like a user interface for computers. Increasingly, companies are required to process their financial reports through APIs. The Securities and Exchange Commission provides an API through their sec.gov website to retrieve the financial reports of companies.

This section will show you how to scrape data from websites and utilize simple API's to collect data.

### ***Web scraping to collect data***

Scraping data from the web consists of two main actions. First, you must connect to a website from within your computer program. This is similar to what the web browser does behind the scenes. This can be accomplished with Python libraries that allow you to make hypertext transfer protocol (HTTP) requests. To make an HTTP request with these libraries, you simply need to know the URL to the website where you want to collect the data. The Python library does the rest for you. We will examine the requests library to make HTTP requests.

Second, you need a library to extract data from the HTML content returned from the HTTP request. To scrape data from the web, you must understand how particular websites structure their data within HTML tags. If you understand the HTML structure, you can use a parser to extract data from that structure. A parser reads the tags within the HTML structure and returns the content within tags when asked to do so. To send HTTP requests, we will use the requests library; for HTML parsing, we will use the BeautifulSoup library.

To get started, install the request and BeautifulSoup libraries. This can be done with the following commands:

```
pip install requests  
pip install beautifulsoup4
```

The requests library is fairly easy to use. You simply need to know the web address of the site you want to scrape. In the example below, the address to the FinViz website is used. FinViz offers free information about corporations, such as their yearly sales, income, Beta values, dividends paid, and various ratios like price-to-earnings ratio and earnings per share. The pandas-datareader library and Alpha Vantage API that you saw in earlier chapters to collect stock price information lack some of these additional details. If you wanted more information about a company and its performance, you could scrape the data from FinViz or other sites like FinViz.

The web address to FinViz for this example is: <https://finviz.com/quote.ashx?t= tsla>. This would retrieve data for Tesla (i.e., TSLA). The “TSLA” at the end of the web address could be replaced with any other stock ticker, such as GOOG or F, to retrieve data for those respective companies. However, you must note that if FinViz changes their website structure or CSS styling, the code will no longer work. You would then need to adjust the patterns the example looks for. If you want to see what the requests library will return, visit the address in a browser, right click on the page, and click the option to view the source code. The code you see in your browser is what the requests library will return.

After knowing the address, you should plan to add “headers” to the request. Many websites expect to find certain “headers” associated with each HTTP request. A header carries information about the request, such as data about the requester. When you use a browser, the headers contain information about your particular browser and/or operating system. We can mimic these expectations by manually adding header information. Header information can be passed through the “headers” parameter of the get() method within the requests library. The example below shows a User-Agent header to mimic what a browser would send. Without the User-Agent, FinViz would not return the requested data. The get() method accepts the http address of the website and any headers. In response, the get() method returns a response object.

The response object contains information about the request, such as whether the request was completed successfully. If errors occurred, the status\_code attribute will provide an error value that most web developers will understand. Status code 200 means that the request was completed successfully. The “ok” attribute of the response object provides a True/False value; True if the status code was 200 and False otherwise. If the request was successful (i.e., response.ok was True), you can extract the HTML code from the response object through the “content” attribute. This is shown in the example below. You are now finished with the requests library.

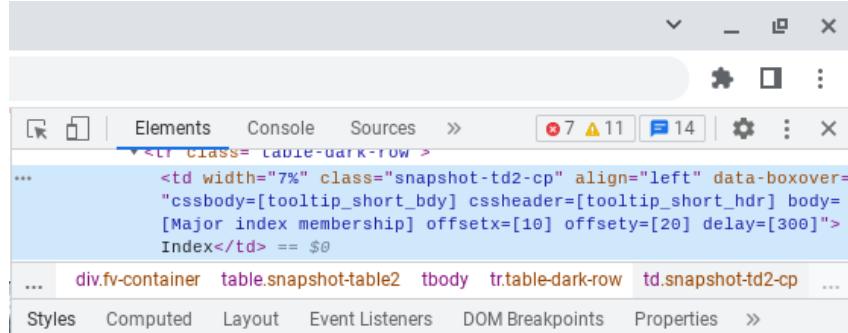
Once you have the raw HTML content from the requests library's response object, you can begin to parse through the HTML to find the specific information that you want. You will do this with the BeautifulSoup library. Often, it is **easiest to find the information you want manually in a browser first** before doing so with BeautifulSoup. Figure 10.1 shows a screenshot of part of the FinViz website at the link provided earlier. The Chrome browser has some useful tools to help you understand the HTML structures surrounding particular information that you wish to extract. In Chrome, if you right click on the information that interests you, such as the word "Index" in Figure 10.1, you can click the "Inspect" option in the dropdown.

**Figure 10.1: Data from the FinViz Website**

Index	S&P 500	P/E	97.30	EPS (ttm)	2.78	Insider Own	0.10%	Shs Outstand	3.11B	Perf Week	-6.21%
Market Cap	<b>868.47B</b>	Forward P/E	<b>46.63</b>	EPS next Y	<b>5.79</b>	Insider Trans	<b>-75.25%</b>	Shs Float	<b>2.62B</b>	Perf Month	<b>-12.45%</b>
Income	<b>9.51B</b>	PEG	<b>1.86</b>	EPS next Q	<b>1.05</b>	Inst Own	<b>44.40%</b>	Short Float	<b>2.30%</b>	Perf Quarter	<b>15.22%</b>
Sales	<b>67.17B</b>	P/S	<b>12.93</b>	EPS this Y	<b>669.20%</b>	Inst Trans	<b>3.71%</b>	Short Ratio	<b>0.71</b>	Perf Half Y	<b>-3.30%</b>
Book/sh	<b>11.69</b>	P/B	<b>23.11</b>	EPS next Y	<b>40.35%</b>	ROA	<b>15.00%</b>	Target Price	<b>298.80</b>	Perf Year	<b>10.68%</b>
Cash/sh	<b>5.89</b>	P/C	<b>45.91</b>	EPS next 5Y	<b>52.28%</b>	ROE	<b>29.80%</b>	52W Range	<b>206.66 - 414.50</b>	Perf YTD	<b>-23.29%</b>
Dividend	-	P/FCF	<b>61.76</b>	EPS past 5Y	<b>48.60%</b>	ROI	<b>15.70%</b>	52W High	<b>-34.81%</b>	Beta	<b>2.20</b>
Dividend %	-	Quick Ratio	<b>1.10</b>	Sales past 5Y	<b>50.40%</b>	Gross Margin	<b>27.10%</b>	52W Low	<b>30.63%</b>	ATR	<b>12.14</b>
Employees	<b>99290</b>	Current Ratio	<b>1.40</b>	Sales Q/Q	<b>41.60%</b>	Oper. Margin	<b>15.90%</b>	RSI (14)	<b>40.51</b>	Volatility	<b>4.20% 4.05%</b>
Optionable	Yes	Debt/Eq	<b>0.12</b>	EPS Q/Q	<b>91.40%</b>	Profit Margin	<b>14.20%</b>	Rel Volume	<b>0.60</b>	Prev Close	<b>277.16</b>
Shortable	Yes	LT Debt/Eq	<b>0.08</b>	Earnings	<b>Jul 20 AMC</b>	Payout	<b>0.00%</b>	Avg Volume	<b>84.85M</b>	Price	<b>270.21</b>
Recom	<b>2.40</b>	SMA20	<b>-7.55%</b>	SMA50	<b>0.09%</b>	SMA200	<b>-9.17%</b>	Volume	<b>50,597,418</b>	Change	<b>-2.51%</b>

The Inspect tool will open on the right side of the browser. Figure 10.2 shows what the Inspect tool looks like. The Inspect tool will show the HTML related to the information that you right clicked on. In this example, that was the word "Index". In Figure 10.2, you will notice that the word Index is contained within a <td> HTML tag. The <td> tag is used to store data within a table. More importantly, the <td> cell has a particular CSS "class" attribute that is distinct to other classes on the FinViz web page. If you click on other information within the table, you will see the same <td> tag and the same class. The data titles have a class named "snapshot-td2-cp" and the <td> tags containing the actual company data have a class named "snapshot-td2". You can exploit patterns like these to extract the information you want from a web page with the BeautifulSoup library.

**Figure 10.2: Inspect Tool in the Chrome Browser**



Once you have identified patterns in the HTML that identify certain data values, you can begin parsing the data with the BeautifulSoup library. Simply call BeautifulSoup() to instantiate a BeautifulSoup object, passing it the HTML content from the requests response object. Since it is HTML, you also need to set the parser to "html.parser" as shown in the example. Once a BeautifulSoup parser object is created, the find\_all() method can be called on the object to find the patterns you identified in the HTML. In the example below, the find\_all() method is passed "td" because the data was stored in <td> tags. The second parameter of the find\_all() method is a dictionary of the HTML tag attributes that match the pattern in the HTML. In the example, the <td> tags had distinct classes (i.e., "snapshot-td2-cp" and

“snapshot-td2”). These values can be added to the dictionary (i.e., “class”: “snapshot-td2-cp”). The `find_all()` method returns an array of HTML objects that meet the criteria. In this example, the criteria was “td” tags with a class set to “snapshot-td2-cp” and another where the class attribute is set to “snapshot-td2”.

The `find_all()` method returns an array with the values collected based on the criteria provided. Because of the HTML structure, the arrays will contain the titles and values of the stock information on the FinViz website. In the example below, this information is then printed using a loop. The result would display something like:

Index: S&P500

P/E: 97.30

EPS (ttm): 2.78

...

An example of using the `requests` and `BeautifulSoup` libraries to scrape data from the web.

```
import requests
from bs4 import BeautifulSoup

def getFinVizTickerData(ticker):
    header = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36'}

    response = requests.get(f"https://finviz.com/quote.ashx?t={ticker}", headers=header)

    return response

def parseFinVizData(response):
    if response.ok:
        parser = BeautifulSoup(response.content, 'html.parser')

        tickerDataTitles = parser.find_all('td', {'class': 'snapshot-td2-cp'})
        tickerDataValues = parser.find_all('td', {'class': 'snapshot-td2'})

        return {"titles": tickerDataTitles, "values": tickerDataValues}

    return False

TSLAResponse = getFinVizTickerData("TSLA")
TSLAData = parseFinVizData(TSLAResponse)

for i in range(len(TSLAData["titles"])):
    title = TSLAData["titles"][i].text
    value = TSLAData["values"][i].text

    print(f"{title}: {value}")
```

Notice in the example above that the text within the HTML tags is extracted with the `.text` attribute. Each tag that can contain data will have a `.text` attribute. Besides extracting text from an HTML tag, you might

also want to extract the values of HTML tag attributes. For example, the `<a></a>` tags are used to create links between web pages. The `href` attribute specifies where the anchor tag will send users. For example, the following tag would send a user to Google's website if they were to click on the text Google:

```
<a href="https://google.com">Google</a>
```

When scraping a website, finding link addresses can be useful. You might want to collect all of the web addresses on a page and parse, send a request to each of those links, and parse the content on each linked page. This could be accomplished using the `requests` library and the `BeautifulSoup` parser. To access an attribute of a tag, such as the `href` attribute, simply reference the attribute name as the key of a dictionary. For example, the code below would find all `<a>` tags and extract the text of the link and the web address (i.e., `href`) of the link:

```
parser = BeautifulSoup(response.content, 'html.parser')
links = parser.find_all('a')

for link in links:
    linkText = link.text #retrieve the text from the <a> tag
    linkAddress = link["href"] #retrieve the href attribute from the <a> tag

    print(f"{linkText}: {linkAddress}")
```

The `BeautifulSoup` library has many other functions that make data collection possible. This example shows some of the core functions, namely `find_all()`. A `find()` method also exists that performs similar behavior, but only returns the first instance of an HTML tag. If you want to learn more about `BeautifulSoup` and web scraping, you can access the `BeautifulSoup` documentation here:

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>.

**You must be cautious in your use of web scraping.** The example above could break at any time if the FinViz website decided to redesign the look of the web page you were scraping. To fix the code, you would have to examine the new HTML and CSS structure to find new patterns for data extraction. Constant monitoring of your web scraper becomes necessary to ensure that it remains a useful tool as websites change. A web scraping tool also becomes useless if a website decides to shutdown.

## **Scraping data within embedded HTML structures**

In the examples above, the information was fairly easy to find because it was contained in an HTML tag (i.e., the `<td>` tag) with distinct classes that weren't used anywhere else in the interface (i.e., `snapshot-td2-cp` and `snapshot-td2`) or all of one type of tag was desired (i.e., all `<a>` tags).

Not all websites are as friendly to scrape as the example shared above, and not all use cases are as easy as finding all instances of a tag. In fact, some websites use Javascript programming to make it nearly impossible to scrape the website. In other cases, the HTML structures simply lack unique CSS classes or other tag attributes that make it easy to find information with a single call to the `find()` or `find_all()` methods.

When the structure of an HTML page has no distinctive CSS classes or other defining tag attributes, you may need to follow the deeply embedded structure of nested HTML tags. Some methods to accomplish this include chaining together `find_all()` or `find()` methods. For example, the following code would help you find a `<div>` tag nested in a `<td>` tag nested in a `<tr>` tag.

```
parser = BeautifulSoup(response.content, 'html.parser')
```

```
data = parser.find('tr').find('td').find('div').text
```

A fair amount of data is contained in HTML tables. Tables are created with <table></table> tags. Rows in the table are created with <tr></tr> tags (i.e., table row tags) that are embedded within the <table> tags. Data cells in each row are created with <td></td> tags (i.e., table data tags). The <td></td> tags represent the columns and the data within the columns. If you have a table with multiple rows, each containing multiple data cells, you might need to parse the table using embedded for loops. In the finviz example, the <td> cells all contained a unique CSS class which made it easy to find them without loops. However, this won't always be the case. In the code below, for-in loops are nested to explore all cells within all rows of every table on the web page.

An example of using loops to find nested tags.

```
import requests
from bs4 import BeautifulSoup

#assumes a response was received from a requests object
parser = BeautifulSoup(response.content, 'html.parser')

#get all tables in the web content
tables = parser.find_all('table')

for table in tables:
    #get rows within each table
    rows = table.find_all('tr')

    for row in rows:
        #get data cells within each row
        dataCells = row.find_all('td')

        for data in dataCells:
            #print the information in each data cell
            print(data.text)
```

With the few methods you have now learned, you can retrieve a variety of information from many different websites. You now have access to massive amounts of information that you didn't before.

As a word of caution, this knowledge can be used unethically. **You should be thoughtful in the way that you use collected data and how you collect it.** When you access data with the requests and BeautifulSoup libraries, it is as if you are visiting the web page with a browser. A program can access a website many times per second via a loop in your program. Humans cannot. This means that you can overload website servers that don't have protections in place. Many websites do have protections against this. If you try to access the same website too many times within a certain time period, the website will block you from accessing their content. Be cautious and responsible when using web scraping technologies.

## **Using API's to collect data**

Application programming interfaces (APIs) allow one computer to request information from another computer. In practice, this is often a program on one server requesting information from a program on

another server. However, you can also request information from another company's server with a notebook on your computer. In fact, you learned to access data from a simple API in Chapter 6. You saw how the pandas-datareader library easily connects to the Yahoo Finance API to collect stock price and volume data. You also saw how to connect to the Alpha Vantage API with a key. Unfortunately, not all tools are as easy to use as pandas-datareader.

In Chapter 7, you learned that databases are often protected by APIs. A great deal of data within databases is accessed through an API. APIs provide a layer of security to manage who can access data in databases and how much data a particular user can access. Complex APIs require several steps to securely connect to the API to make requests to retrieve data. This section is meant as a brief introduction to APIs and will not cover advanced API connection and authentication frameworks. However, this section will show you how to connect to simple APIs that are more complicated than the pandas-datareader library.

Like web scraping, APIs are often accessed through libraries that can make HTTP requests. In this section, you will see examples that utilize the Python requests library. This is the same library used to access data for web scraping. APIs are often accessed through web addresses. However, unlike the web addresses you visit with a web browser, the data provided by an API is not typically in HTML format. Instead, APIs return data in a computer readable format, such as extensible markup language (XML), Javascript object notation (JSON), or protobuf. These formats allow a computer to parse through the data easily. JSON and protobuf are the formats that are currently recommended. XML is an older API communication language. In the next sections, you will learn to connect to an API and parse JSON formatted data.

## ***Connecting to an API***

In this section, you will learn how to connect to the Securities and Exchange Commission's (SEC's) financial reporting API. The SEC's financial reporting API is used to collect financial reports from companies and disseminate financial reports to interested investors. Through the API, you can access the quarterly financial reports (i.e., 10-Q reports) and annual financial reports (i.e., 10-K reports) of publicly traded companies. Other types of reports are also available, but we will focus on the 10-Q and 10-K reports.

API's consist of different "endpoints." **API endpoints** are web addressed where certain types of information can be requested. The SEC API does not require advanced user authentication and authorization mechanisms like many APIs. It is free to use as long as you do so responsibly. The following website describes the privacy and security policies for accessing the SEC API endpoints:

<https://www.sec.gov/privacy>

The actual SEC API endpoints are described on this website:

<https://www.sec.gov/edgar/sec-api-documentation>

The SEC uses a special format for financial data called the extensible business markup language (XBRL). Although originally an XML-based language, the SEC endpoints now provide the XBRL language in JSON format. XML looks similar to HTML except that the tags are custom tags that define the type of information stored within them. The XBRL language breaks financial reports into standardized tags, such

as <AccountsPayableCurrent></AccountsPayableCurrent>, <Assets></Assets>, <CostOfGoodsSold></CostOfGoodsSold>, etc. The SEC API now converts these tags into JSON format to provide more modern API formatting standards. You will see examples of how the XBRL structures JSON data in the next section. For now, let's examine how some of the SEC endpoints can be accessed with the Python requests library.

Importantly, the SEC XBRL API only provides data for companies that submit their quarterly and annual reports in XBRL format. Increasingly, companies are required to use this format, but many older reports cannot be accessed through the SBRL API endpoints. In the example below, you will see how to access all of the XBRL reports for an organization and how to access specific information from financial reports for a specific organization. The SEC keeps track of companies through a unique id called the central index key (CIK). For example, Microsoft's CIK is: 0000789019. If you Google "Microsoft CIK number," you can verify this information. You can find the CIK for any publicly traded company in this manner. You can also use the CIK lookup tool provided by the SEC at: <https://www.sec.gov/edgar/searchedgar/cik>.

In the example below, the requests library is used to connect to the following two endpoints:

- <https://data.sec.gov/api/xbrl/companyfacts/CIK#####.json> - collect all XBRL report data for a company
- <https://data.sec.gov/api/xbrl/companyconcept/CIK#####/us-gaap/AccountsPayableCurrent.json> - collect specific data (e.g., AccountsPayableCurrent) from reports for a specific company

An example of connecting to the SEC endpoints

```
import requests

#collect all financial data from all XBRL reports for a specific company
def getAllReportData(CIK):
    header = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36'}

    #call the API endpoint for the particular CIK passed into the method
    response = requests.get(f"https://data.sec.gov/api/xbrl/companyfacts/CIK{CIK}.json",
                           headers=header)

    return response

#collect specific financial data from all XBRL reports for a specific company
def getSpecificReportData(CIK, dataType):
    header = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36'}

    #call the API endpoint for the particular CIK passed into the method
    response =
        requests.get(f"https://data.sec.gov/api/xbrl/companyconcept/CIK{CIK}/us-gaap/{dataType}.json",
                     headers=header)
```

```

return response

#get all of the financial data for all XBRL reports from Microsoft (i.e., CIK 0000789019)
AlljsonData = getAllReportData("0000789019")

#get the diluted earnings per share for all XBRL reports from Microsoft (i.e., CIK 0000789019)
EPSjsonData = getSpecificReportData("0000789019", "EarningsPerShareDiluted")

```

The code in the two methods above should look familiar. It is similar to the requests you saw for web scraping. The main difference is that web scraping requests return HTML code from a website, while XBRL API requests return JSON code from an API endpoint that is not intended for human consumption. JSON code looks similar to Python dictionaries. JSON consists of key:value pairs.

In order to utilize the second method (i.e., getSpecificReportData()), you must know how companies use XBRL labels to identify different types of financial data. For example, the diluted earnings per share value on Microsoft's XBRL report is stored with the key EarningsPerShareDiluted. The “undiluted” earnings per share is stored with the key EarningsPerShareBasic, gross profits with the key GrossProfit, etc. In the next section, you will learn how to parse through the information returned from the API endpoints to make sense of it and find the data you want.

### **Parsing JSON data**

JSON stands for Javascript Object Notation. **JSON** represents a way to share information contained with object-oriented programming objects as a String data type. You cannot pass Python objects or objects from other programming languages over a network, such as the Internet. To pass information stored in objects, you must convert it to an encoded String (i.e., XML or JSON) or convert the object data to binary format (i.e., protobuf). Extensible markup language (XML) was the default method to share data over a network, however, it lacked a clear relationship to object-orientation. JSON was created with object-orientation in mind. It has quickly become the default for modern data sharing over APIs. Protobuf is a newer technology that is starting to be utilized more, but is not the dominant method for data sharing.

JSON consists of key:value pairs much like Python dictionaries. Just like Python objects can contain other objects or lists, JSON can store embedded objects or lists of information within objects. You can read more about JSON at: <https://www.json.org/json-en.html>. Like the Python notation for dictionaries, JSON objects are contained with curly brackets (i.e., {} ) and stored in key:value pairs. An example can be viewed at: <https://json.org/example.html>.

Let's assume you have built a system that allows a financial manager to manage their clients. In the system, the ClientManager object would contain multiple Client objects which could each contain multiple Portfolio objects. The JSON might look something like the code below. The outer curly brackets represent the ClientManager object. The word “clients” represents the attribute clients of the ClientManager which contains multiple Client objects (i.e., Jamie Doe and Ryan Doe). Each Client object has one or more Portfolio objects (i.e., Jamie's Retirement Account and Education Account and Ryan's Family Trust).

```
{
  "clients" : [
    {"name" : "Jamie Doe", "phone" : "555-555-5551",

```

```

    "portfolios" : [
        {"name" : "Retirement Account", "value" : 23465.12},
        {"name" : "Education Account", "value" : 2924.56}
    ],
},
{"name" : "Ryan Doe", "phone" : "555-555-5553",
 "portfolios" : [
    {"name" : "Family Trust", "value" : 143763.39}
]
}
]
}

```

To retrieve data in JSON format, you will need to import the json Python library. Once imported, you can pass a JSON code through the loads() method of the json library. This will convert the JSON code into a Python dictionary for you to utilize. You can then access information from the dictionary in the way you learned in previous chapters. The code below shows how you would convert the JSON code above into a Python dictionary and access the first Portfolio object of the Client “Ryan Doe.”

Parse JSON data for use in a program

```

import json

#assume you have created a function named collect() that returns JSON
#assume the jsonResponse looks like the JSON presented earlier
jsonResponse = collect()

#if you printed the jsonResponse, it would look like the JSON presented earlier
print(jsonResponse)

#use the json library to convert the JSON to a Python dictionary for use in the program
data = json.loads(jsonResponse)

#print information contained within the dictionary
print(data["clients"][1]["portfolios"][0]["name"])

#The above line would print out "Family Trust" because that is the portfolio name of the 1st portfolio
#(i.e., the portfolio at index 0) of the second client (i.e., the client Ryan Doe at index 1).

```

Using the library above, you can parse any correctly formatted JSON response returned from an API and utilize the data via a Python dictionary. For example, you could build on the XBRL example in the previous section by passing the EPSJSONData variable into the json.loads() method to extract the information in the JSON string into a data structure usable within the Python program. If you collect the correct information from the SEC API, you could calculate a variety of ratios that represent the different perspectives on the performance of an organization.

## Parsing financial data from the SEC's API

The JSON output of the SEC's company facts API returns a great deal of information. The response contains all 10-Q reports (i.e., quarterly financial reports) and 10-K reports (i.e., annual financial reports) for a company since the company started filing XBRL reports in JSON format. It can feel overwhelming when you first glance at the JSON output of the API. The following is the first part of the JSON output for Microsoft:

```
{'cik': 789019, 'entityName': 'MICROSOFT CORPORATION', 'facts':....}
```

The outer object of the JSON response shown above contains details about the company, such as its CIK number and company name. The “facts” attribute of the company stores fact objects based on different accounting taxonomies established by the SEC. For example, the “dei” taxonomy shown below represents the SEC’s Document and Entity Information (DEI) taxonomy. This taxonomy provides a great deal of information about the company from phone numbers to details about company stock shares. Visit this site for more information about the DEI taxonomy: <https://xbrl.us/xbrl-taxonomy/2021-dei/>.

```
{'cik': 789019, 'entityName': 'MICROSOFT CORPORATION', 'facts': {'dei':
```

Another important SEC taxonomy you will see regularly is “us-gaap.” US GAAP stands for the US Generally Accepted Accounting Principles. Visit this site for more information about the US GAAP taxonomy: <https://xbrl.us/xbrl-taxonomy/2021-us-gaap/>. The US GAAP sets forth requirements for companies as they report their quarterly and annual financial reports. The US GAAP describes what type of financial data needs to be reported quarterly and annually, and how the data should be labeled to maintain consistent reporting standards across companies. The JSON output below shows two fact objects, namely the dei facts and us-gaap facts.

```
{'cik': 789019, 'entityName': 'MICROSOFT CORPORATION', 'facts': {'dei': {...}, 'us-gaap': {...} } }
```

Within each SEC taxonomy are concepts. These concepts represent specific data required within the financial report. For example, the DEI taxonomy for Microsoft returns the EntityCommonStockSharesOutstanding concept as the first concept in the output. Each concept possesses a number of important attributes. The “label” provides a human readable label for the data represented by the concept. The “description” attribute describes the concept in English. The “units” attribute defines the unit of analysis for the concept. In the example below, the units is “shares” because the concept is EntityCommonStockSharesOutstanding. Another common unit is USD, which stands for US dollars.

```
{'cik': 789019, 'entityName': 'MICROSOFT CORPORATION', 'facts': {'dei':  
{'EntityCommonStockSharesOutstanding': {'label': 'Entity Common Stock, Shares Outstanding',  
'description': "Indicate number of shares or other units outstanding of each of registrant's classes of capital or common stock or other ownership interests, if and as stated on cover of related periodic report. Where multiple classes or units exist define each class/interest by adding class of stock items such as Common Class A [Member], Common Class B [Member] or Partnership Interest [Member] onto the Instrument [Domain] of the Entity Listings, Instrument.", 'units': {'shares':
```

The particular concept units, such as “shares” or “USD”, contain a list of concept report objects as reported on 10-Q and 10-K reports from the time the company started reporting in XBRL format. These report objects are where the values for the concept are stored. Each report listed in the units object represents details provided for that concept on a specific 10-Q or 10-K report filing. In the example below, the EntityCommonStockSharesOutstanding concept shows up on multiple reports. Two of those reports are shown below: one for the first quarter 10-Q report submitted in 2010 and another for the second

quarter 10-Q report submitted in 2010. Each of these concept report objects contains a series of important attributes. The val attribute contains the actual value that shows up on the report for the particular concept. The fy attribute stands for fiscal year and the fp attribute stands for fiscal period. The form attribute presents which type of report was filed (i.e., 10-Q or 10-k).

```
{'cik': 789019, 'entityName': 'MICROSOFT CORPORATION', 'facts': {'dei': {'EntityCommonStockSharesOutstanding': {'label': '...', 'description': '...', 'units': { 'shares': [{'end': '2009-10-19', 'val': 8879121378, 'accn': '0001193125-09-212454', 'fy': 2010, 'fp': 'Q1', 'form': '10-Q', 'filed': '2009-10-23', 'frame': 'CY2009Q3I'}, {'end': '2010-01-25', 'val': 8770460922, 'accn': '0001193125-10-015598', 'fy': 2010, 'fp': 'Q2', 'form': '10-Q', 'filed': '2010-01-28', 'frame': 'CY2009Q4I'}]}
```

To retrieve the appropriate data from these JSON outputs, you must first understand how the data is structured. The SEC website provides updated details about different accounting reporting taxonomies. With the information shared above, you possess a basic understanding of the structure of the SEC's XBRL JSON outputs. Using the json library, you can convert this JSON data to a Python dictionary and begin analyzing the health of organizations.

For example, if you want to print out the EntityCommonStockSharesOutstanding for Microsoft for each quarter provided, you would need to access the the ["facts"] dictionary, then the ["dei"] fact dictionary, then the ["EntityCommonStockSharesOutstanding"] concept dictionary, then the ["units"] dictionary and ["shares"] dictionary. You would then receive a list of reports of common stock shares outstanding.

Parse SEC JSON data for use in a program

```
import requests
import json

#collect data from the SEC company facts API
header = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36'}

response = requests.get("https://data.sec.gov/api/xbrl/companyfacts/CIK{CIK}.json", headers=header)

if response.ok:
    #convert the JSON response to Python dictionaries
    data = json.loads(jsonResponse)

    #get the common shares outstanding reports from the embedded dictionaries
    commonSharesOutsandingReports =
        data["facts"]["dei"]["EntityCommonStockSharesOutstanding"]["units"]["shares"]

    #loop over each report and print the shares outstanding information for each quarter
    for report in commonSharesOutsandingReports
        print(f"The shares outstanding on the {report['form']} report filed for {report['fp']} of {report['fp']} are {report['val']}")

    #if you wanted to know which US GAAP concepts were included in the reports you could do:
    gaapConcepts = data["facts"]["us-gaap"]

    for concept in gaapConcepts
        print(f'{data["facts"]["us-gaap"][concept]["label"]}: {data["facts"]["us-gaap"][concept]["description"]}'")
```

With this information, you can begin to analyze financial reports in a more automated fashion. If you aren't sure what concepts are included in the XBRL output, you can loop over the ["facts"]["dei"] or ["facts"]["us-gaap"] dictionaries. You can then print out the label and description of each concept in the data. An example of this is shown at the end of the example above.

## Advanced Data Preprocessing

Web scraping often results in the collection of non-numeric data. Many emerging business data sets include a variety of non-numeric data, such as: text, images, videos, sound, etc. Even traditional business data sets include non-numeric data mixed with numeric data. For example, a sales dataset might include the name of a salesperson. Similarly, a data set with the cost and quantity of assets owned by the organization might record the asset's category. Stock market data can include information about the industry of a company. You will encounter these and other non-numeric data in traditional business data sets. Advances in machine learning allow for non-traditional, non-numeric data as well. For example, the voice data from customer service call centers can be processed and analyzed. Visual data from cameras can be converted to numeric form for analysis. Textual information from historic reports can be processed and analyzed as well. These and many other non-traditional forms of data are being processed in ways that were previously inaccessible.

To begin, let's examine how to process non-numeric data found in traditional data sets. Data can be collected using different levels of measurement. Depending on the type of measurement, you will need to prepare the data in specific ways. Four primary levels of data measurement exist: nominal, ordinal, interval, and ratio.

### ***Levels of measurement***

**Nominal** measurement scales are the simplest form of measurement and are textual in nature. Nominal scales are used for data that does not contain any inherent order. For example, SIC codes, which designate the industry a company belongs to, do not offer any information about the importance of a particular industry in relation to another industry. Nominal data is a type of categorical data. SIC codes simply represent different industry categories that can be assigned to companies. Although nominal data is textual in nature, you can assign numbers to represent each category in the scale. This process will be described later. An example of a nominal scale is provided below. Notice that the values have no inherent order. They could be listed in any order without changing the meaning of the measurement.

Example of a nominal scale:

Population Designation: ["Rural", "Suburban", "Urban"]

**Ordinal** measurement scales are categorical and textual in nature as well. However, unlike nominal scales, ordinal scales contain some inherent order in the measurement scale. For example, you could subjectively rate the risk of a business decision as high, medium, or low risk. Although this scale is textual and lacking information about the distance between the categories, the order of the scale matters. Low risk is clearly less risk than medium risk, which in turn is less risk than high risk. However, we don't know what the difference between low risk and medium risk is. There is no inherent distance between low, medium, high. Ordinal data can be converted to numerical data by assigning sequential values to the different categories (e.g., Low=1, Medium=2, High=3). Likert scales (e.g., 1-5 or 1-7 agree to disagree

scales) are ordinal scales converted to numeric scales. However, Likert scales are often treated as interval scales.

Example of an ordinal scale:

Population Designation: ["Low Population", "Moderately Low Population", "Moderate Population", "Moderately High Population", "High Population"]

**Interval** measurement scales are numerical in nature. Interval scales have inherent numeric order (e.g., 2 is greater than 1) and the distance (i.e., interval) between numbers is meaningful. For example, the distance between three and four is the same as the distance between four and five. Although interval scales have numeric order and meaningful intervals, they do not have a "true" zero. In order to exhibit a true zero, the value zero must be meaningful to the scale. Let's take the common example of temperature in Fahrenheit to explore this idea further. The Fahrenheit scale is numeric, the interval between degrees is meaningful (e.g., the distance between 30 and 60 degrees is the same as the distance between 60 and 90 degrees). However, zero degrees Fahrenheit does not mean the absence of heat. Therefore, zero in Fahrenheit is not a true zero.

**Ratio** measurement scales are numerical in nature as well. Ratio scales have inherent numeric order, meaningful intervals, and a true zero. The existence of a true zero allows for the creation of ratios. For example, a city with a population of 200 people has twice as many people as a city with a population of 100 people. The Kelvin temperature scale has a true zero. Zero degrees in Kelvin means the absence of heat. Thus, Kelvin is a ratio scale, while Fahrenheit is an interval scale. In many statistical methods, interval and ratio data are treated similarly.

Although interval and ratio data can be used in their raw form within many statistical analyses, nominal and ordinal data must be converted to numeric form.

### **Converting nominal data to numeric form**

**Nominal data can be converted into numeric form in two ways.** The simplest method is to assign a sequential integer value to each category in the nominal scale. For example, suppose you had the following categories for Bachelor's degrees: MIS, Accounting, and Finance. With the first method, you would simply assign: MIS=1, Accounting=2, Finance=3. **One problem with this method is that it treats nominal data that has no inherent order as though it did.** Based on the conversion, some statistical methods would see Finance as having a greater value than Accounting and MIS. However, this is not true for nominal data.

**To overcome this weakness, the second method creates multiple new columns (i.e., features) called dummy variables.** In each new column, the data would be converted to binary 0 or 1 values. For example, two new columns would be created for the example above: BachelorMIS, BachelorAccounting, BachelorFinance. We would not need to use the third column in many statistical analyses, because it provides redundant information. For example, if a particular record had a 0 in the BachelorMIS and BachelorAccounting columns, then it would be known that the student majored in Finance. The third column would provide redundant information, which can be problematic for certain types of analyses.

The table below shows the two different ways to convert nominal data to numeric data. The column Bachelor is the raw nominal data. The BachelorNumeric column would represent the first problematic method. The three columns BachelorMIS, BachelorAccounting, and BachelorFinance would be created

for the second method. Notice the first row. The student is MIS. In the first method, a value of 1 would be assigned to it. In the second method, the BachelorMIS would be set to 1 (i.e., True) and the two other dummy variables would be set to 0 (i.e., False) for the row. This pattern continues for all of the other rows.

StudentId	Bachelor	BachelorNumeric	BachelorMIS	BachelorAccounting	BachelorFinance
1	MIS	1	1	0	0
2	Accounting	2	0	1	0
3	Finance	3	0	0	1
4	MIS	1	1	0	0
5	MIS	1	1	0	0

Both methods can both be accomplished with relative ease using the scikit-learn Python library. The first method can be accomplished by using the LabelEncoder class built into scikit-learn. The second method, called one-hot encoding, can be accomplished by using the get\_dummies() method in pandas or the OneHotEncoder class built into the scikit-learn. The get\_dummies() method supports data frames better than the OneHotEncoder library, which relies on numpy arrays. First, import the library you want to use to encode the data. In the example below, both methods are shown for illustration. The one hot encoding method is accomplished with the pandas get\_dummies() method. The import statements are:

```
from sklearn.preprocessing import LabelEncoder
import pandas as pd
```

The code below uses a data set obtained from Kaggle.com, which is a site where individuals can share datasets and submit machine learning analyses on the data sets. If you want to work with the particular data set, visit: <https://www.kaggle.com/datasets/lava18/google-play-store-apps?resource=download>. At the time of this writing, Kaggle offered free accounts. This particular data set consists of data on more than 10,000 apps on the Google Play Store. The data set has many nominal features, such as app category and content rating. For simplicity, the content rating feature is encoded since it has fewer categories (i.e., 'Everyone', 'Teen', 'Everyone 10+', 'Mature 17+', 'Adults only 18+', 'Unrated'). Figure 10.3 shows the first few rows of some of the data set columns.

**Figure 10.3: Sample Data with Nominal Columns**

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone	Art & Design	January 7, 2018	1.0.0	4.0.3 and up
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+	Free	0	Everyone	Art & Design;Pretend Play	January 15, 2018	2.0.0	4.0.3 and up
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+	Free	0	Everyone	Art & Design	August 1, 2018	1.2.4	4.0.3 and up
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25M	50,000,000+	Free	0	Teen	Art & Design	June 8, 2018	Varies with device	4.2 and up
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2.8M	100,000+	Free	0	Everyone	Art & Design;Creativity	June 20, 2018	1.1	4.4 and up

The code below would convert the Content Rating using the two different methods (i.e., sequential integers vs. one hot encoding).

Example of using label and one hot encoding for nominal data

```
from sklearn.preprocessing import LabelEncoder
import pandas as pd

data = pd.read_csv("googleplaystore.csv")

#instantiate a LabelEncoder to use the first method (i.e., sequential integers) to encode the data.
encoder1 = LabelEncoder()

#create a new column in the data frame and fit_transform the Content Rating column using the encoder
data["ContentRatingMethod1"] = encoder1.fit_transform(data["Content Rating"])

#the new column would look like this
```

	Content Rating	ContentRatingMethod1
0	Everyone	1
1	Everyone	1
2	Everyone	1
3	Teen	4
4	Everyone	1

```
#to use one hot encoding on a data frame (i.e., the second method), call the get_dummies() method
#the prefix parameter adds a prefix to the new column names
#the columns parameter designates which columns need to be converted to dummy variables
data = pd.get_dummies(data, prefix=["Rating"], columns=["Content Rating"])
```

#the output of the change would look like this:

ContentRatingRaw	Rating_Adults only 18+	Rating_Everyone	Rating_Everyone 10+	Rating_Mature 17+	Rating_Teen	Rating_Unrated
Everyone	0	1	0	0	0	0
Everyone	0	1	0	0	0	0
Everyone	0	1	0	0	0	0
Teen	0	0	0	0	1	0
Everyone	0	1	0	0	0	0

## Converting ordinal data to numeric form

Converting ordinal data to numeric is much simpler than converting nominal data. The pandas replace() method described in the data analytics chapter can be used to make ordinal data numeric. Depending on how you planned to use the Content Rating column, you could argue that Content Rating is ordinal. If you were to interpret Content Rating as the restrictiveness of the rating, then Content Rating could be interpreted as ordinal. In this case, the order of the ratings would be (from least to most restrictive): 'Everyone'=1, 'Everyone 10+'=2, 'Teen'=3, 'Mature 17+'=4, 'Adults only 18+'=5. The 'Unrated' data could be set to null or encoded as the absence of a rating (i.e., 0). In the example below, the 'Unrated' data is set to null. The code below shows how the replace() method can be used to alter ordinal data to numeric data.

Example of using replace() to convert ordinal data to numeric data.

```
import pandas as pd
import numpy as np

data = pd.read_csv("googleplaystore.csv")

#create a new column with a more appropriate description for ordinal data.
data["RatingRestrictiveness"] = data["Content Rating"]
```

```

#set the rating to null (i.e., np.nan) if the rating is "Unrated"
data["RatingRestrictiveness"] = data["RatingRestrictiveness"].apply(lambda rating: np.nan if
rating=="Unrated" else rating)

#drop rows with null values with the dropna() method
data = data.dropna()

#convert the content labels to numbers using the replace() method with a dictionary
data["RatingRestrictiveness"] = data["RatingRestrictiveness"].replace({'Everyone':1, 'Everyone 10+':2,
'Teen':3, 'Mature 17+':4, 'Adults only 18+':5})

```

#the new column would look like this

	Content Rating	RatingRestrictiveness
0	Everyone	1
1	Everyone	1
2	Everyone	1
3	Teen	3
4	Everyone	1

## Text Preprocessing

The previous section demonstrated how to convert nominal and ordinal data to numeric data. Not all textual data is treated in a traditional sense as nominal or ordinal. Natural language processing is concerned with examining textual data, often without any other numerical variables. When you wish to process large textual documents, such as what you might find when data scraping a website, a different type of data preprocessing is needed. When treating unstructured textual data as a data set, you will want to carefully parse the textual data. Although it still must be converted to numeric form, the process is not the same as converting nominal and ordinal data. The textual data must be normalized, stop words removed, the data tokenized, and then stemmed or lemmatized. Other advanced forms of preprocessing can also be done. Once the data is preprocessed, it must be converted into a form that can be analyzed, such as a document-term matrix or word embedding.

Python contains useful libraries to preprocess and analyze textual data, such as the natural language toolkit library (i.e., nltk). If not already installed on your computer, the nltk library can be installed in Anaconda Prompt or the Linux/Mac terminal with the command:

**pip install nltk**

The Kaggle data set mentioned earlier contains a second data set with app review text. The Translated\_Review column contains the review text. This data will be referenced below. Figure 10.4 shows what this second data set looks like:

**Figure 10.4: Sample Textual Data Set**

	App	Translated_Review	Sentiment	Sentiment_Polarity	Sentiment_Subjectivity
0	10 Best Foods for You	I like eat delicious food. That's I'm cooking ...	Positive	1.00	0.533333
1	10 Best Foods for You	This help eating healthy exercise regular basis	Positive	0.25	0.288462
2	10 Best Foods for You		NaN	NaN	NaN
3	10 Best Foods for You	Works great especially going grocery store	Positive	0.40	0.875000
4	10 Best Foods for You		Best idea us	Positive	1.00
					0.300000

### **Normalizing text**

First, you will want to make all of the text look similar by making all of the text lowercase or uppercase and removing punctuation, such as: periods, question marks, commas, semicolons, colons, etc. If you don't remove capital letters, the word "Hello" and "hello" would be treated as two different words. Similarly, if you don't remove punctuation, the word "end." and "end!" would be treated as two different words when they clearly are not.

In a pandas data frame, **converting all text to lowercase** is fairly simple; call the lower() method on the str attribute associated with the String data in a pandas data frame. Or simply call the lower() method on a string if your data is not contained in a pandas data frame.

**To remove punctuation**, the replace() method can be used. Python has a built-in string library that has a punctuation attribute that returns common punctuation. This library can be imported to make it easier to remove punctuation from text using the replace() method. First, the punctuation from the string library must be converted to a list using the list() method. Then, you can loop over the punctuation and perform a str.replace() method for each different punctuation mark. Replace the punctuation with nothing (i.e., "").

### **Tokenizing text**

Once you have textual data that is fairly uniform (i.e., no punctuation and all text in lowercase or uppercase format), you can begin to extract individual words from the text. This process is known as **tokenization**. In natural language processing, the term **token** is often used to refer to a single word or a word fragment. Thus, to tokenize a text document is to break the text into the unique words within the text.

Tokenization can be accomplished in a few different ways. First, string data has a built-in method named: split(). The split() method will break down a text document into a list of individual words by looking for spaces between words. Using the split() method will retain every instance of each word. The nltk library also has a tokenization feature. You must download the nltk tokenization feature using the download() built into the nltk library. The name of the tokenization feature to download is "punkt". Unlike the split() method, the nltk tokenization method does not retain each instance of a word. Instead, it keeps only the first instance of each word. If a word is repeated in a text document, the nltk tokenizer will only keep one instance of a word.

### **Removing stop words**

Once the textual data is normalized by converting the text to lowercase or uppercase and removing punctuation, and the text is tokenized, it is important to remove regular words, called stop words. **Stop**

**words** are commonly used words like: the, and, or, that, this, don't, etc. If a word is too common, it is unlikely to be of value in an analysis.

Stop words can be removed by using a downloadable feature in the nltk library. Stop words do not come pre-installed when the nltk library is installed. The download() method of nltk can be used to download the stopwords feature: `nltk.download("stopwords")`. The nltk stopwords list contains apostrophes, which were removed from the textual data when normalizing the text. Thus, the apostrophes must be removed from the stopwords as well by using the same method to remove punctuation. If you don't remove the apostrophes, you cannot compare the stop words to the normalized text because the words with apostrophes won't match anything in the normalized text. Alternatively, the stop words could be removed before removing punctuation from the text. Once you have a list of stop words, you can retain only the words from your tokenized text that are not included in the stop words list.

## **Stemming and lemmatizing text**

**Stemming and lemmatization** are methods to reduce words to their root form. For example, the words kick, kicked, kicking, kicks, etc. have the same root word (i.e., kick). These different variants of the root word create unnecessary differences between words that have similar meanings, which increases the sparsity of the data set. Stemming and lemmatization can help to increase the density of data.

Stemming and lemmatization approach reducing words to their root in distinct ways. **Stemming works by removing prefixes or suffixes from words.** Unfortunately, this can sometimes lead to over stemming words. Over stemming refers to two distinct words that have very different meanings, yet both are reduced to the same word stem. A common example of over stemming is the reduction of university, universe, and universal to univers. These are distinct words that are reduced to a single stem. Lemmatization is more thoughtful than stemming. **Lemmatizers use a vocabulary and possess an understanding of how words morph within a language.** Words are carefully mapped to a root that is an actual word. For example, the word "better" would be mapped to "good" with a lemmatizer, whereas a stemmer would simply try to trim prefixes and suffixes from the words creating two distinct stems for better and good.

The nltk has easy to use libraries to complete stemming or lemmatization. You only need to complete one or the other. Lemmatization is often the preferred method.

## **Converting text to a data set**

Statistical models do not read text; they require numbers. After preprocessing the text in a document or group of documents, you must convert all of the text to numeric format. There are many ways to do this. For example, you can create an index for each unique word in a text corpus (i.e., group of text documents). These numeric indices can be used to replace the words themselves.

Some textual analyses require that you create a document-term matrix. The document-term matrix is a two-dimensional data structure that contains all of the terms in the text corpus along the columns of the matrix and the text documents along the rows. Within the cells of the matrix are the frequencies of the occurrence of each term within each document. The table below shows a simple document-term matrix. Each column represents a different term. Each row represents a text document, such as a distinct customer product review. The numbers in the cells represent how many times a particular word was referenced in the document.

**Example of a Document-Term Matrix**

	apple	eat	build	tree	hat	pear	hot	good	done
review1	2	1	0	1	0	0	0	0	0
review2	0	0	0	2	0	1	2	1	0
review3	0	0	2	0	1	0	1	0	0
review4	0	0	1	0	0	0	0	2	3

Notice in the matrix above that the data contains many 0's. This data is known as sparse, because very few cells have non-zero values. Sparse data sets are less efficient than dense data sets. As such, some advanced models rely on denser textual data sets. These denser data sets are sometimes called word embeddings. Word embeddings are created with advanced statistical methods that are beyond the scope of this text.

## ***Text preprocessing in Python***

Python can be used to perform the various textual preprocessing steps. Preprocessing can be accomplished in a customized fashion using the nltk library or by using a simpler Python class in the sci-kit learn library.

The code below shows how the various aspects of text preprocessing can be paired together with customized methods. Notice that the natural language toolkit library (i.e., nltk) is used extensively in the example. This library has many tools for textual analysis to help you preprocess and analyze textual data simply. The nltk library is large, as it contains many textual vocabularies. As such, many of the nltk features must be downloaded after importing the library using the download() method. If you forget to download a feature that you need to perform a particular task, you will receive an error message with the name of the feature that you need to download.

Using the nltk library to create customized text preprocessing methods

```
import string
import nltk
import pandas as pd
from nltk.stem import WordNetLemmatizer

nltk.download("stopwords")
nltk.download('wordnet')
nltk.download("punkt")

def getTextFromDFColumn(data, columnName):
    #join data from the text column into one variable
    text = " ".join(data[columnName])
    return text
```

```

def lowercaseText(text):
    #make all of the text lowercase
    text = text.lower()
    return text

def removePunctuation(text):
    #remove punctuation from text using the string library
    punctuation = list(string.punctuation)

    for mark in punctuation:
        text = text.replace(mark,"")

    return text

#break text string into individual words. Can remove duplicate words or retain all words
def tokenizeText(text, removeDuplicates=False):
    if removeDuplicates:
        #tokenize and remove duplicate tokens
        terms = nltk.word_tokenize(text)
    else:
        #tokenize and retain duplicate tokens
        terms = text.split()

    return terms

def removeStopWords(terms, textHasPunctuation=False):
    #get the dictionary of common English stop words
    stopWords = nltk.corpus.stopwords.words('english')

    if not textHasPunctuation:
        #remove apostrophes from stop words so they match text with punctuation removed
        for i in range(len(stopWords)):
            stopWords[i] = stopWords[i].replace("'", "")

        terms = [term for term in terms if term not in stopWords]

    return terms

def lemmatizeTerms(terms, removeDuplicates=True):
    lemmatizer = WordNetLemmatizer()
    lemmatizedTerms = [] #create data structure for terms with duplication
    lemmatizedTermSet = set() #create data structure for terms without duplication

    #pass each term through the lemmatizer to get the lemma from each word
    for term in terms:
        if removeDuplicates:
            lemmatizedTermSet.add(lemmatizer.lemmatize(term))
        else:
            lemmatizedTerms.append(lemmatizer.lemmatize(term))

    if removeDuplicates:
        lemmatizedTerms = list(lemmatizedTermSet)

```

```

return lemmatizedTerms

#using the methods to create a list of preprocessed terms

#get the data from a csv file, remove null values, and reset the index of the data frame
data = pd.read_csv("googleplaystore_user_reviews.csv")
data = data.dropna()
data = data.reset_index()

#retrieve and preprocess the text using the methods created above
text = getTextFromDFColumn(data, "Translated_Review")
textLowercased = lowercaseText(text)
textNoPunctuation = removePunctuation(textLowercased)
tokens = tokenizeText(textNoPunctuation)
tokensNoStopWords = removeStopWords(tokens)
tokensLemmatized = lemmatizeTerms(tokensNoStopWords)

print(tokensLemmatized)

```

If you need a document-term matrix and don't want to worry about the details of the way the textual data is preprocessed, you can rely on the CountVectorizer module within the sci-kit learn library. Notice that the CountVectorizer library requires much less programming. It is possible to add further customization to the CountVectorizer, such as implementing a specific lemmatizer. However, the base library provides text normalization features, removal of stop words, tokenization, etc. The document-term matrix produced by the CountVectorizer can be converted to an array or to a data frame.

For large data sets, you can also reduce the number of term features used in the document-term matrix. For example, text from the Google Store reviews takes more than 6GB of memory to store the full document-term matrix. Most laptops are not equipped with that much memory. When conducting analyses on a simple computer, it may be necessary to reduce the number of terms used in the document-term matrix. The max\_features parameter of the CountVectorizer limits how many terms to include in the matrix. The code example below shows how to use the CountVectorizer

Using the CountVectorizer module to create a document-term matrix

```

import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

#get the data from a csv file, remove null values, and reset the index of the data frame
data = pd.read_csv("googleplaystore_user_reviews.csv")
data = data.dropna()
data = data.reset_index()

#instantiate the CountVectorizer
vectorizer = CountVectorizer(stop_words="english", max_features=1000)

```

```
#convert the text column into a document-term matrix using fit_transform()
matrix = vectorizer.fit_transform(data["Translated_Review"])
```

```
#get the list of terms from the vectorized data
columns = vectorizer.get_feature_names()
```

```
#convert the matrix to array form and then to a pandas data frame
df = pd.DataFrame(matrix.toarray(), columns=columns)
display(df)
```

The output would look like this with shape (37,427 documents, 1000 terms):

	10	100	12	15	18	20	2018	24	30	50	...	worth	wow	write	writing	wrong	year	years	yes	youtube	zero
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
37422	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	0	0	0	0
37423	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
37424	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
37425	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
37426	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

Depending on your needs, you can now quickly create a textual data set with the CountVectorizer or create your own custom text preprocessing methods.

With an increased knowledge about data collection and cleaning tools and practices, you can begin analyzing multitudes of data to make better managerial decisions.

## Chapter 10 Assignment

Please see the [Chapter Assignments Workbook](#) for the Chapter assignment.

# Chapter 11:

## Introduction to Machine Learning

Machine learning is one of the core technologies underlying the automation that is being implemented and will continue to be implemented in industry over the next several decades. Machine learning is a branch of artificial intelligence that seeks to teach computers to learn to act and make decisions. Machine learning moves away from many of the concepts described in previous chapters, such as hard coded flow control logic. Instead, machine learning seeks to provide systems with adaptive intelligence. Many advances have taken place in the study and practice of artificial intelligence and machine learning, and many more will take place over the next decade. You should understand the fundamentals of machine learning.

Although complex machine learning models, such as neural networks are becoming increasingly common, these large and complex models are not always needed. Many different machine learning models exist. Some models, such as cluster analysis, are unsupervised learning models. Other models, such as decision trees and support vector machines, are supervised learning models. As a reminder, supervised models require labeled data sets where the outcomes of each data record are known. Unsupervised models do not require labels. Instead, the model finds patterns and the data analyst or data scientist labels the data after the analysis.

This chapter will introduce you to several basic machine learning models. The focus of this chapter will be to provide a conceptual understanding of the models. The underlying mathematics of the models are not discussed deeply in this text. Statistical or machine learning texts provide details about the underlying mathematics of these models if you are interested in learning more about how they work. Understanding the conceptual assumptions and limitations of the models is enough for you to begin utilizing them. Many of the models can be implemented with only a few lines of Python code. Work with the statisticians and data scientists in your organization to build new and interesting automation tools for business.

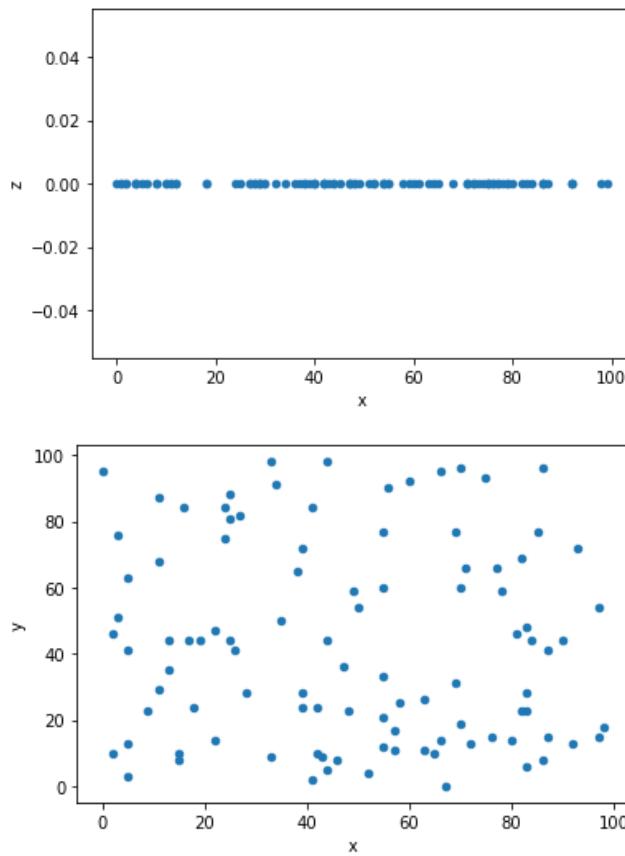
### Dimensionality Reduction

With an understanding of how to collect data and clean it in simple and advanced ways, you are now prepared to learn about dimensionality reduction. Dimensionality reduction is common in analytics and machine learning analyses. To understand dimensionality reduction, you must first understand what “dimensionality” and the “curse of dimensionality” refer to. **Dimensionality** simply refers to the number of features in a data set. For example, you have seen the Alpha Vantage data set multiple times. The data set has the following features: High, Low, Open, Close, and Volume. The data set has five features, and therefore, five dimensions. To graph the interactions between the five features/dimensions, you would need to be able to graph in a five-dimensional space. In terms of visualizations, we are limited to three dimensions. However, data can exist in far more dimensions than we are able to visualize. Even though our visual abilities are limited to three dimensions, mathematics does not possess such limitations.

The “**curse of dimensionality**” refers to the issue that occurs with data as the dimensionality of a data set increases. As dimensionality increases, the sparsity of the data increases as well. **Sparsity** refers to an increase in space between data points. The curse of dimensionality can be visualized by comparing one dimensional and two dimensional plots, each with the same number of data points. In Figure 11.1,

you will see two plots. The first is a one-dimensional plot with one hundred randomly generated data points. The distance between the points is minimal. The second plot is a two-dimensional data set with one hundred randomly generated data points. As you can see, there is far more white space between the points in the two-dimensional plot than in the one-dimensional plot. A three-dimensional plot with one hundred data points would look even more sparse. If four-, five-, and six-dimensional plots were even possible, you would notice the data becoming increasingly sparse. To make up for this sparsity, you often need far more data to find meaningful patterns and relationships in the data.

**Figure 11.1: One- and Two-Dimensional Plots Showing Sparsity**



## Principal component analysis

If you have a data set with 200 features, you are working within a 200-dimensional space. Unless you had a very large dataset, the data would likely be sparsely distributed throughout the 200-dimensional space making certain analyses difficult. Fortunately, it is possible to reduce the number of dimensions in a data set by removing or combining data features while maintaining the quality of the data. Although many dimensionality reduction techniques exist, this chapter introduces you to a common dimensionality reduction technique: **principal components analysis** (PCA).

PCA reduces the dimensionality of a data set by calculating linear combinations from the data feature values using weight vectors called principal components. A **principal component** is a weight vector, called an eigenvector, calculated from the covariance matrix of the data set that can be used to transform data into a smaller dimensional space. Each principal component represents one dimension. The idea is

to calculate and use less principal components than data features. If you had 200 features and could reduce that to 50 principal components, you would move from a 200-dimensional space to a 50-dimensional space. In some cases it is possible to reduce the dimensionality even further without reducing the quality of the data too much.

We will not examine how eigenvectors are calculated. Mathematics textbooks provide greater detail about how PCA eigenvectors are calculated. Optimization calculations are required to identify the eigenvectors, which is beyond the scope of this text. We will simply explore how eigenvectors can be computed by Python analytics libraries and used to create a data set with fewer dimensions. The example below shows how a weight vector would be applied to raw data to reduce the dimensionality of the data set.

### An example of PCA

Suppose you are examining stock data from the pandas-datareader library or Alpha Vantage API that has the following features: High, Low, Open, Close, Volume. Suppose the raw price and volume data looked like this:

	High	Low	Open	Close	Volume
2020-01-08	216.24	212.61	213.00	215.22	13475000
2020-01-09	218.38	216.28	217.54	218.30	12642800
2020-01-10	219.88	217.42	219.20	218.06	12119400
2020-01-13	221.97	219.21	219.60	221.91	14463400
2020-01-14	222.38	218.63	221.61	219.06	13288900

Notice in the data above that the values in the Volume column are much larger than the values in the other columns. Because PCA examines variance and covariance, the analysis can be influenced by differences in the relative size of feature values. Variance tends to be greater for features with larger values. Because Volume has much larger values than the price data, the Volume feature would receive greater weight in the PCA analysis if raw data were used. However, there is no intuitive reason why Volume should receive greater weight simply because its values are orders of magnitude greater than the price data. To overcome this issue, it is best to standardize the data before conducting PCA. You learned in an earlier chapter that the z-score is a common form of data standardization. To begin, convert the raw data to z-scores to put all features on the same scale. This could be accomplished with a lambda function that applies the z-score formula or through a Python library that does the mathematics for you (e.g., <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>). The example below uses a lambda function with the z-score formula:

```
data.apply(lambda x: (x - x.mean()) / x.std())
```

The resulting data table would look something like this:

	High	Low	Open	Close	Volume

2020-01-08	-1.38	-1.61	-1.60	-1.37	0.31
2020-01-09	-0.54	-0.21	-0.20	-0.09	-0.62
2020-01-10	0.04	0.23	0.31	-0.19	-1.21
2020-01-13	0.86	0.91	0.43	1.42	1.42
2020-01-14	1.02	0.69	1.05	0.23	0.10

Notice that the relative size of the Volume feature is not overly exaggerated after standardizing the data, meaning the variance will also be less exaggerated. This standardization process is sometimes referred to as **normalization**. This normalization process centers the data around a mean of 0, which is helpful in the PCA analysis.

Once the data is scaled, a weight vector (i.e., eigenvector or principal component) is then calculated by the PCA algorithm in a way that maximizes the variance explained within the data. Although you can utilize multiple principal components (i.e., eigenvectors), we will start by looking at just one principal component. If you did not normalize the data, the following eigenvector might be produced, which clearly favors the Volume column by giving it a weight of 1.0 compared to the other weights that are near 0.0.

High	Low	Open	Close	Volume
0.00000085	0.00000053	-0.00000002	0.00000128	1.000000000

By standardizing the data, the eigenvector is not so heavily influenced by the large values of the Volume feature. The standardized data would produce the following eigenvector:

High	Low	Open	Close	Volume
-0.50460724	-0.50804382	-0.47762119	-0.48040018	-0.1683944

Notice that each of the standardized data columns now has a new weight. Only the Volume column decreased; all other weights increased as compared to the raw data weights. Once the weight vector is calculated, the values of each row are multiplied by their respective weight and then added together (i.e., a linear combination) to create a **component score**.

Let's examine what this would look like for the first row of the standardized data. The five values in the first row will be reduced to one value (i.e., component score) through the operation. In the data below, the first row is the first row from the standardized data set. The second row is the weight vector calculated from the standardized data set (i.e., the same data as shown above). The third row is the standardized values multiplied by the respective weight value.

High	Low	Open	Close	Volume	
-1.38	-1.61	-1.69	-1.37	0.31	
*	-0.50460724	-0.50804382	-0.47762119	-0.48040018	-0.16839440

$$= \begin{matrix} 0.69635799 & 0.81795055 & 0.76419390 & 0.65814825 & -0.05220226 \end{matrix}$$

When ***the weighted values are added together***, the result is a weighted component score that represents a single value to represent the five values in the first row of the data set. In this case, the component score would be 2.884448 (with some rounding error) for the first row of raw data. You could then repeat this process for each row of data (i.e., multiplying each row's values by the weight vector and then adding the weighted values together) to create component scores for each row. By doing so, you would have reduced your dataset from five features to one feature. The resulting table of component scores would look like this:

	First Principal Component: Component Scores
2020-01-08	2.88958712
2020-01-09	0.62471892
2020-01-10	0.00957619
2020-01-13	-2.02691826
2020-01-14	-1.49696398

In this simple example, you saw a raw data set with five features reduced to component scores using the first principal component (i.e., eigenvector) of the standardized data set. The data above could now be used in an analysis to represent the five columns in the original data set with only one dimension.

The principal component that you saw is called the **first principal component**. It is the principal component that explains the most variance in the data. Although it explains the most variance, it may not always explain all or even most of the variance in the data set. To improve the amount of variance explained by PCA, you often want to utilize multiple principal components (i.e., multiple eigenvectors). Each component represents a new dimension in the reduced data set. Importantly, you don't want to include too many principal components or the analysis does not reduce the dimensionality enough to make it worthwhile. Although you can technically have as many components as features, this is not useful in practice. You want the least number of components that capture a reasonable amount of the variance in the data.

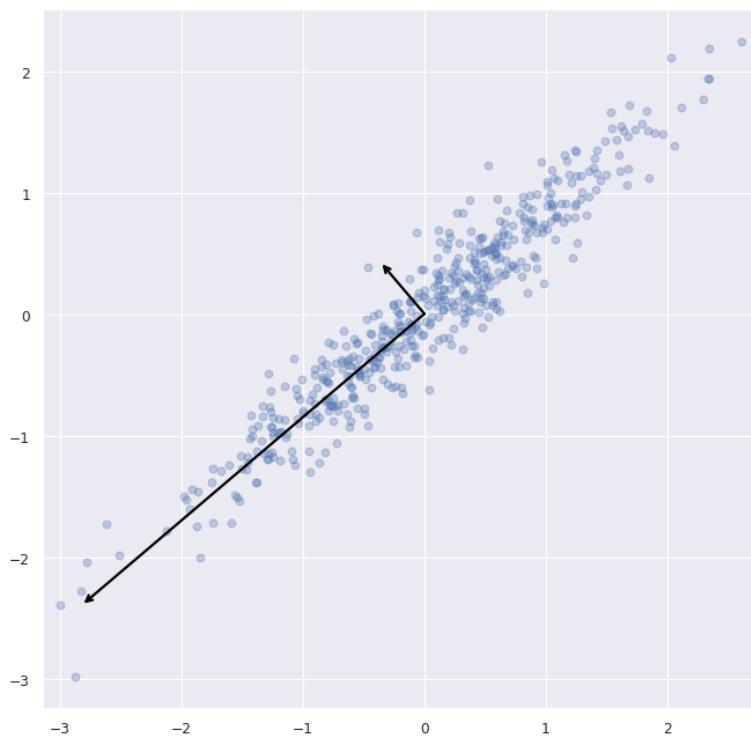
### Geometric representations of principal components

Again, the first principal component is the vector that explains the most variance in the data (linearly speaking). The **second principal component** is the eigenvector that explains the greatest amount of variance **that has yet to be explained** by the first component. To create this second principal component, the second eigenvector is made orthogonal to the first eigenvector. **Orthogonal** is a mathematical term that geometrically speaking means perpendicular. In terms of geometric space, an eigenvector (i.e., weight vector) and its related eigenvalue (i.e., amount of variance explained) is simply a line that points in a specific direction and has a specified length. The length of the eigenvector (i.e., the eigenvalue) is determined by how much variance the eigenvector explains. The first principal component is the longest vector as it explains the most variance.

Intuitively, a line orthogonal to another line will explain the next greatest amount of unique variance (i.e., uncorrelated variance). As the slope of the second line approaches the slope of the first line, the amount of new variance explained would decrease because the non-orthogonal second line would share an increasing amount of variance with the first line and less unique variance. The third principal component (i.e., eigenvector) would also be orthogonal. In terms of geometrical space, a third feature dimension would be required to allow for an orthogonal third component.

Figure 11.2 shows the first two principal components plotted over the top of standardized data. Notice in the figure that the first principal component is the longest line and clearly has the best fitting linear relationship with the data points. The second principal component is orthogonal to the first to maximize additional variance explained.

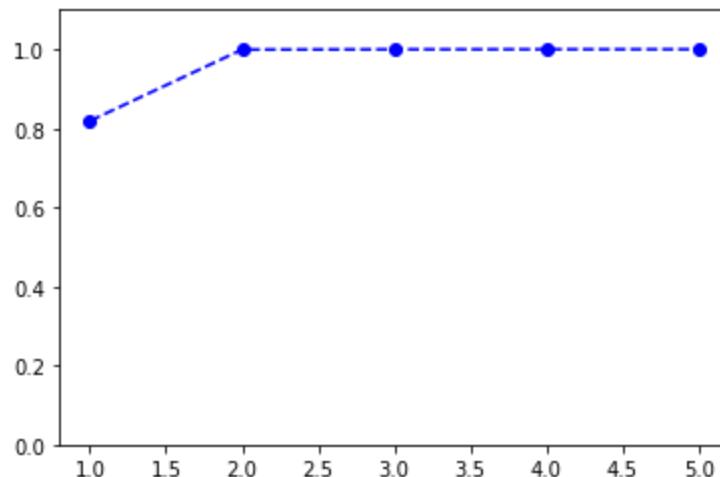
**Figure 11.2: First Two Orthogonal Principal Components**



**You must choose how many principal components to include in your analysis.** There are a number of ways to choose the number of principal components to retain in your data set. First, you can set a threshold for the amount of variance you wish to explain. Reducing dimensions reduces the amount of variance you can explain. Although reducing dimensionality is useful, doing so at the expense of your ability to explain variance in the data is not helpful. Some analysts will set a **threshold** like retaining the ability to explain 95 percent of the variance in the data. Other analysts rely on scree plots which graph the amount of additional variance explained by each component. A heuristic approach to interpreting scree plots suggests that you choose the number of components at the “elbow” in the plot, where the amount of variance explained by the next principal component is relatively small. These plots are sometimes called elbow plots.

You can also plot the cumulative amount of variance explained by each subsequent component. Figure 11.3 shows such a plot with the stock price and volume data presented earlier. Again, a cutoff value can be used, such as 95 percent, to select the number of principal components you would retain. The plot below shows that two principal components would capture nearly 100 percent of the variance. In this case, only the first two components would need to be retained to retain a reasonable amount of variance. This would reduce the dimensionality of the data from five to two dimensions. Although this example is small, big data sets in industry can contain thousands of dimensions. In such cases, PCA or other feature reduction techniques can be particularly helpful.

**Figure 11.3: Plotting Cumulative Variance Explained to Determine Number of Components**



## PCA with Python

Now that you have a conceptual understanding of how PCA works, let's learn to conduct an analysis in Python using the scikit-learn library (i.e., `sklearn`). The scikit-learn library provides you with easy implementations for a number of machine learning models, such as PCA, cluster analysis, decision trees, and support vector machines. We will explore some of these models in later sections of this chapter. For now, let's examine some code to figure out how to conduct a principal components analysis. You will see that with very little code, you can perform the tasks described above with relative ease. To begin, you must install scikit-learn. This can be done with pip in the Anaconda Prompt or in the Terminal and Mac/Linux machines.

### **pip install scikit-learn**

Once installed, you will need to import the PCA module into your notebook with the import statement:  
`from sklearn.decomposition import PCA`

After importing the PCA module, you must create a PCA object. If you instantiate a PCA object with no arguments, the PCA will contain all possible principal components. You can also use the `n_components` parameter to set the number of principal components to use in the analysis. The `n_components` parameter can be set with integer values representing a specific number of components. Alternatively, it can be set with decimal values representing a threshold for the amount of variance you wish to explain (i.e., 0.95). The library will then choose the lowest number of components that are above the set threshold. Once you create a PCA object, you can call the `fit()` method to calculate the eigenvectors and

eigenvalues. The fit() method must be passed a data set as an argument. The data set can be a pandas data frame or a multidimensional numpy array. If you wish to calculate a new data set consisting of the principal component scores, the transform() method must be called. Alternatively, the fit\_transform() method can be called instead of separate fit() and transform() method calls.

Example of using scikit-learn to conduct a principal components analysis.

```
from sklearn.decomposition import PCA

data = pd.DataFrame() #the data frame would go here. This is just an empty data frame

#transform the raw data to z-scores: (feature value - mean of feature) / standard deviation of feature
standardizedData = data.apply(lambda x: (x - x.mean()) / x.std())

#set the desired threshold for the variance explained (i.e., 0.95) or the number of components
pca = PCA(n_components=0.95)
#could also be PCA(n_components=5) or PCA()

#calculate the eigenvectors and eigenvalues of the components
pca.fit(standardizedData)

#create a new data set with reduced dimensionality (i.e., principal component scores)
newData = pca.transform(standardizedData)

#alternatively fit_transform() could have been used instead of fit() and transform()
#newData = pca.fit_transform(standardizedData)
```

Some statistical and machine learning models utilize PCA to reduce the dimensionality of a data set before feeding the reduced data into the machine learning model. Interestingly, PCA can also be used for more than just dimensionality reduction. When using PCA for dimensionality reduction, you try to preserve as much variance in the data as possible by retaining the principal components that explain the most variance. However, you can use this model in reverse to identify anomalies, such as in time series data or image data. By focusing on the principal components with the least variance, you can find anomalous patterns. PCA is a powerful tool that can aid you in your data analytics endeavors.

## Cluster Analysis

Cluster analysis is an unsupervised learning technique that allows you to identify patterns in data without the need for labeled data sets. Cluster analysis is an exploratory analysis approach; it seeks to find patterns in data. The analyst must then make sense of what the patterns mean (i.e., label the patterns). For example, cluster analysis is used in marketing to identify different types of customers based on similarities between data records. Although cluster analysis could identify different groups of customers based on feature values, it would be up to the analyst to label the groups the analysis found (i.e., loyal customers older than 50 who purchase regularly vs. loyal customers older than 50 who purchase semi-regularly vs. non-loyal customers older than 50, etc.).

Cluster analysis creates clusters of related data records based on the features in the data set by examining different measures of distance between data values in dimensional space, the density of data

points, the distribution of data (e.g., Gaussian normal distributions), etc. Cluster analysis is not one single type of analysis, but a series of related analyses.

## **Common types of cluster analysis**

Many types of cluster analysis exist. The main difference between many of the different types is how each type identifies clusters (i.e., groupings) of related data records. Three types of cluster analysis we will discuss include centroid-based approaches, density-based approaches, and distribution-based approaches.

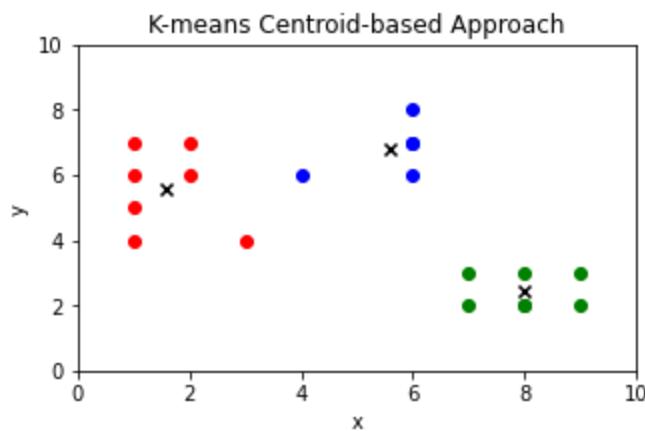
### **Centroid-based clustering**

**Centroid-based clustering** approaches seek to identify clusters by calculating the distance between data points in a multi-dimensional space (based on the number of features used in the analysis) in relation to a certain number of centroids. Each data point is assigned to the centroid closest to it. Each centroid and the data records around it represents a cluster. **Centroids** are points in the dimensional feature space that best separate the data into distinct groups. You can think of centroids as the mean of the data that make up each cluster identified by the clustering algorithm. The k-means clustering algorithm is an example of a centroid-based clustering approach. K-means clustering is discussed more in the next section. Documentation for the K-means method in the sci-kit learn library can be found here:

<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html?highlight=kmeans#sklearn.cluster.KMeans>

Figure 11.4 shows the centroids of a data set marked as x's and the associated data points around each centroid (i.e., the cluster). The data points surrounding each centroid are colored differently to show the distinct clusters. The graphic depicts a two-dimensional feature space (i.e., x and y). However, many data sets have far more dimensions. It becomes impossible to visualize these data sets simply, hence the use of two-dimensions.

**Figure 11.4 Visualization of Centroid-based Clustering**



### **Density-based clustering**

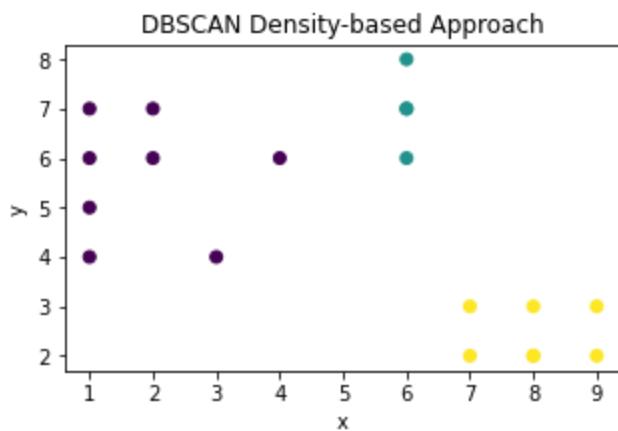
**Density-based clustering** approaches identify clusters within data by calculating the density of data points within the multi-dimensional feature space. Density is a measure of how close data points are to

other data points. In many data sets, some data points are densely clustered in different parts of the dimensional space, while others data points are sparsely distributed in the space. Space between densely clustered data defines boundaries between different data clusters. The DBSCAN clustering algorithm is a commonly used example of a density-based clustering algorithm. Documentation for using the DBSCAN method can be found here:

<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html>.

Figure 11.5 shows an example of density-based clustering. In the figure you will notice that the data points in each colored cluster are densely packed together. The space between the dense clusters acts as the separating boundaries between the three dense clusters.

**Figure 11.5: Visualization of Density-Based Clustering**



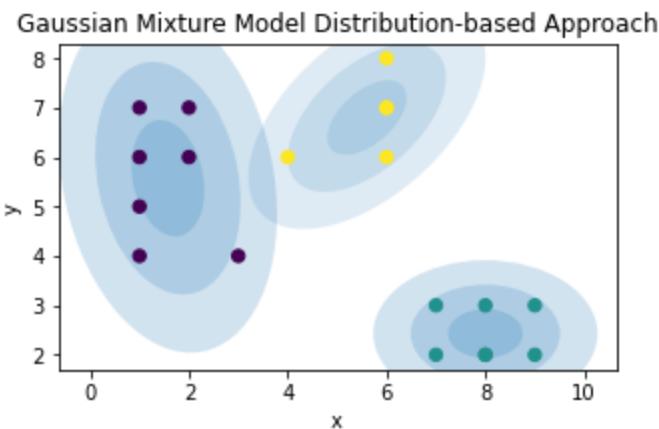
### **Distribution-based clustering**

**Distribution-based clustering** approaches identify clusters within data by calculating distributions for the data, such as Gaussian distributions (i.e., bell curves). Data that fit within different distributions are paired into different clusters. The analyst decides how many distributions (i.e., components) to break the data into. Because the analysis is based on different distributions, you can calculate the likelihood that any given data point belongs to a particular distribution. Points closer to the center of the distribution (e.g., mean) are more likely to belong to that distribution. Points farther from the center of the distribution are less likely to belong to the distribution. Some data points can overlap between two distributions, making this clustering approach a “softer” clustering method. In models like KMeans, each point belongs to only one cluster. Because distribution models are probabilistically based on distributions, you can calculate probabilities for data points near the border of two overlapping distributions. Gaussian mixture models are a common form of distribution-based clustering analysis. You can learn more about Gaussian mixture models with the sci-kit learn library here:

<https://scikit-learn.org/stable/modules/mixture.html#gaussian-mixture>

Figure 11.6 shows Gaussian distributions surrounding data points. The points closer to the darker circles of each cluster have a higher likelihood of belonging to that distribution. Points in the lighter circles are less likely to belong to the distribution. As shown by the distribution to the left and the distribution near the center, distributions can overlap.

**Figure 11.6: Visualization of Distribution-based Clustering**



### **Which clustering model to use**

Choosing which clustering model to use requires expertise. Many analysts use the k-means clustering approach in practice, largely because it is the simplest to understand and implement. Centroid-based approaches rely on fairly simple calculations of mean distance between data points and centroids, whereas density-based approaches rely on measures of density and distribution-based approaches rely on multiple measures such as mean and variance. The simplicity of the centroid-based k-means algorithm is what makes it so popular. Still, you should consult a data scientist or statistician in your organization to determine which type of clustering model to use based on the nature of the data.

Although the next section highlights the k-means algorithm and how to implement it in Python, you should recognize that choosing the simplest method for an analysis is not always the best choice. This is not to say that you should not use the k-means clustering algorithm. Many use it and find meaningful relationships that help to make decisions about how to treat different customers, employees, etc. Because of its simplicity, it can also act as an early prototype analysis to determine if time and energy should be expended in conducting more complex analyses. This is true for many types of models. Neural networks in their various forms can be costly and time consuming to build. If there are no clear relationships in the data, even neural networks will provide little value. You cannot find meaningful patterns where no meaningful patterns exist. As such, you can often choose simpler models to determine if patterns exist in data before spending time and money developing more complex models that provide additional predictive or explanatory accuracy.

If you wish to learn more about which type of clustering model is best for different types of data, you will want to read texts specifically about cluster analysis or take a mathematics or computer science course focused around cluster analysis. The Internet also has many resources on specific clustering algorithms. This section is meant only as an introduction to the topic.

### **K-means clustering**

The k-means clustering algorithm is a centroid-based clustering approach. The analyst selects a number of centroids and fits the data to the specified number of centroids. The "k" in the k-means name refers to the number of centroids selected by the analyst. Again, a **centroid** represents the mean of a cluster of

data, hence the “means” part of the k-means name. K-means simply refers to the multiple (i.e., k) means (i.e., centroids) used in the analysis.

In the simplest implementation of the k-means algorithm, the location of each of the k centroids in the multi-dimensional feature space is randomly seeded. Once the centroids are randomly seeded, the distance between each data point and the k centroids is calculated. Each data point is then assigned to the centroid closest to the data point. Then, the location of each centroid is recalculated based on the data points assigned to it. The centroid is moved to the center (i.e., mean) of the data surrounding it. This process is repeated until the mean of each centroid does not change. The following YouTube video shows what this repetitive process looks like graphically. The colored dots shown in the video represent the changing centroid means over the various iterations of the k-means algorithm.

<https://www.youtube.com/watch?v=nXY6PxAaOk0>.

Because the k-means algorithm starts with randomly seeded centroid locations, the results will be different if you run the algorithm multiple times. For this reason, the algorithm is typically run multiple times to find the best fitting centroids for the data set.

You might now be asking yourself, how do you know how many centroids to include in the analysis? The answer to this question is important, as the number of centroids (i.e., k) changes how the data clusters together. Two centroids will give you different results than five centroids. To find the appropriate number of centroids, analysts will often calculate a range for k, such as between 2 and 10 centroids. For each value of k, an error score can be calculated. If you continue increasing the number of clusters, but the error changes very little, you know that you have found a good value for k. You want the fewest number of centroids that provide the greatest reduction in error. A more complex method relies on something called silhouette scores instead of a simple measure of error. We won’t cover these methods in depth, but the code to calculate the ideal k is shown below.

Thankfully, Python makes calculating the k-means algorithm simple through the KMeans module in the scikit-learn cluster library. You can also calculate the ideal value for k using the same library. You must ensure that the scikit-learn library is installed on your computer. Then, you can import the library using:

```
from sklearn.cluster import KMeans
```

After importing the library, you simply need to instantiate a KMeans object and fit it to your data using the fit() method similar to the way you did for the PCA analysis. When instantiating the KMeans object, you can specify the number of centroids to use by setting a value for the n\_clusters parameter. Many of the scikit-learn libraries offer the same simple methods to perform an analysis. Namely, the fit() method and the transform() method, and the fit\_transform() method to execute both at the same time. This is true for the KMeans library as well.

Once the KMeans model is fit to the data (i.e., the centroids are calculated) with the fit() method, you can analyze the outcomes of the analysis, such as determining the location of the centroids. The centroid locations can be identified with the cluster\_centers\_ attribute. The labels\_ attribute can be called to identify which cluster each data point belongs to. The inertia\_ attribute provides a measure of the amount of error in the fitted model.

By creating a loop that iterates over values of k within a certain range (i.e., 2-10), you can perform multiple KMeans analyses and compare error scores from the inertia\_ attribute. You can then model an

elbow chart as depicted in the example below to determine how many centroids to use. The number of centroids can then be set with the KMeans constructor, such as in the example below using the n\_clusters parameter.

```
model = KMeans(n_clusters=3)
```

The KMeans algorithm also performs each cluster analysis repeatedly. Because centroids are randomly distributed through the dimensional space, results can differ if the analysis is repeated multiple times. To improve the likelihood of finding the best location for centroids, the KMeans module will run the analysis repeatedly with different random initializations and find the centroid locations with the lowest error score. You can change how many times the analysis is performed with the n\_init parameter of the KMeans constructor. The default number of iterations is 10. If you want more or less iterations than this, you would simply change the constructor to look like the example below. In the example below, the number of iterations is set to 25 with 3 centroids (i.e., n\_clusters).

```
model = KMeans(n_clusters=3, n_init=25)
```

Example of using scikit-learn to conduct a k-means cluster analysis.

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

#get some data
av = AlphaVantageConnector("AlphaVantageAPIKeyGoesHere")
av.getDaily("GOOG")
data = av.toDataFrame() #convert the AlphaVantage data to a pandas data frame

#create data features to cluster around
X["DayRange"] = data["High"] - data["Low"]
X["GainLoss"] = data["Open"] - data["Close"]

#before running the analysis, find an appropriate value for k
#below is the code to create an elbow plot

#set up a list to store error metrics (i.e., sum of squared error) for each potential value of k
error = []

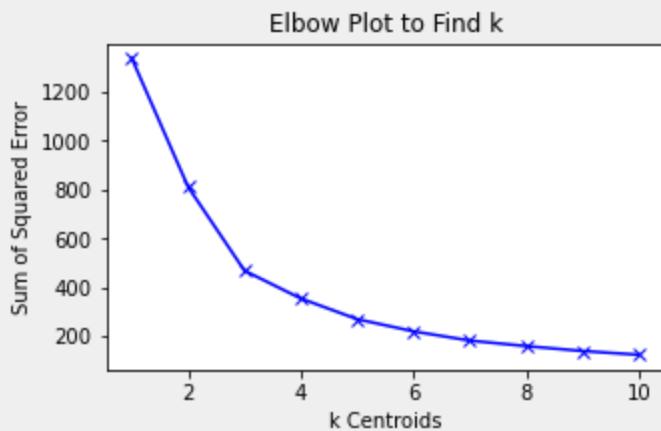
#choose the range of k to explore (i.e., 1-10 centroids)
K = range(1,11)

#for loop over each value of k and fit the model
for k in K:
    #create a k means analysis for each value of k
    model = KMeans(n_clusters=k)
    model.fit(X)

    #add the error (i.e., inertia_) to the error list for the particular value of k
    error.append(model.inertia_)
```

```
#plot the error data
plt.plot(K, error, 'bx-')
plt.title('Elbow Plot to Find k')
plt.xlabel('k Centroids')
plt.ylabel('Sum of Squared Error')
plt.show()
```

#the output would look something like this:



#Notice that the sum of squared error metric stops dropping rapidly after 3 centroids  
#This provides a simple assessment for what value of k you should use

#Once you determine a good value for k, run the k-means for that value of k

```
#instantiate the KMeans model with 3 cluster centroids (i.e., k=3) based on the prior analysis
model = KMeans(n_clusters=3)
```

```
#find the location of the centroids
model.fit(data)
```

```
#print out the location of the centroids
print(model.cluster_centers_)
```

```
#plot each data point and which cluster it belongs to
plt.title('K-means Clustering of Stock Data')
```

```
#set up colors for the data belonging to the two different clusters
clusterColors = ["blue", "green", "red"]
```

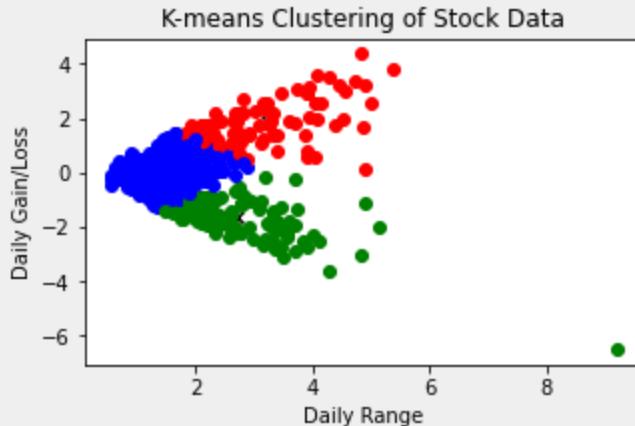
```
#loop over the cluster labels assigned to each data point to display each data point
for i, l in enumerate(model.labels_):
```

```
    #plot the ith data point for each feature. Set the color based on the color list above
    plt.plot(X["DayRange"][i], X["GainLoss"][i], color=clusterColors[l], marker="o")
    plt.xlabel("Daily Range")
    plt.ylabel("Daily Gain/Loss")
```

```
#plot the centroids based on the two feature dimensions
plt.scatter(centers[:,0], centers[:,1], marker="x", color='black')

plt.show()
```

#this code would output something like:



In the example above, the three clusters show three different outcomes for a trading day for this stock. The blue cluster shows that days with small gains/losses will typically have low ranges (i.e., high price - low price). Here the ranges are a very simple measure of volatility within a single trading day. The red cluster shows that days with greater gains are likely to have larger ranges. The green cluster shows that days with greater losses are also likely to have larger ranges. In many ways, this should seem intuitive. Days with greater volatility ranges are going to be more likely to end with greater gains or losses. Said in the opposite way, days with greater gains and losses will display greater volatility throughout the day. In some ways, this logic is circular. Confirming the obvious through an analysis is not particularly helpful.

This raises the question, **what features should you even use in an analysis in the first place?** The simple answer to that is: you need to consult a subject matter expert (e.g., accountants, engineers, etc.). Whether you are leading a team of analysts or acting as an analyst, you are not always going to be an expert at every problem you encounter in business. Good leaders and team members know when to seek expertise. The biggest problem with data analytics is that discernable patterns exist everywhere. If you look hard enough in data, you will eventually find a relationship. The question then becomes, **is the relationship meaningful or valuable?** A data scientist or data analyst may not be able to answer this question alone. As you begin working with data in your future career, be sure to consult subject matter experts to find what features are meaningful and to identify if your patterns are meaningful. Finding patterns is actually fairly easy, as shown above. Finding meaningful patterns that help individuals make decisions is more difficult.

## Supervised Classification Models

Supervised learning models use a data set with features and known outcomes to identify how patterns in the features relate to the outcomes. Often, supervised learning models map the features to outcomes by using a weight matrix, similar to the eigenvector concept you encountered in PCA, or by selecting splitting

criteria for features that best explain the outcomes. When weight matrices are used, the model adjusts weight values until feature values map accurately to the known outcomes. When splitting criteria are used, the model identifies which features best explain the outcomes and identify splitting points within those features. Many neural networks map features to outcomes with weight matrices. Decision trees and random forests rely on splitting criteria.

Supervised models can be used for classification problems or regression problems. In classification problems, the outcomes are classes. For example, you could create a supervised learning model to predict whether you should hire or not hire a potential candidate. The classes within the model might be “hire,” “possibly hire,” and “do not hire.” For each candidate, the model would classify the candidate into one of the three classes. Similarly, classification models are used for object recognition in computer vision models. The classes would represent the different objects (i.e., people, cars, trees, street signs, etc.) that the model could predict from an image. Self-driving vehicles for example use classification models in this way.

Other problems can be modeled as regression problems in which the outcome is a continuous value instead of a finite set of classes. In regression models, you attempt to predict a specific numeric value based on a set of feature values. Some problems can be modeled in both ways. For example, the hiring example could be formulated in a way that a fitness score is calculated for each job candidate instead of a simple set of classes. Some supervised learning models, such as decision trees can be used for both classification and regression problems. For most problems and learning models, classification models are simpler. Since this is an introductory section, we will focus on classification problems.

The process of mapping features to outcomes in a supervised learning model is called **training**. The labeled data used to update the weight matrices or selection criteria that map features to outcomes is called the **training data set**. The `fit()` method is used for most of the supervised learning models built into the sci-kit learn library to use training data to train a supervised model. Once you finish training a supervised model with the training data, you will often want to test how well the model predicts data from a new data set. This data set used for testing is called the **test data set**. It is important to test a trained model on new data because models can learn the particular fine-grained patterns within a training data set (i.e., **overfitting the data**) instead of learning high-level patterns that may exist across data sets.

In the sections that follow, you will be introduced to decision trees, random forests, and support vector machines. Each of these machine learning models is a supervised model that requires training on a training set and testing on a test set.

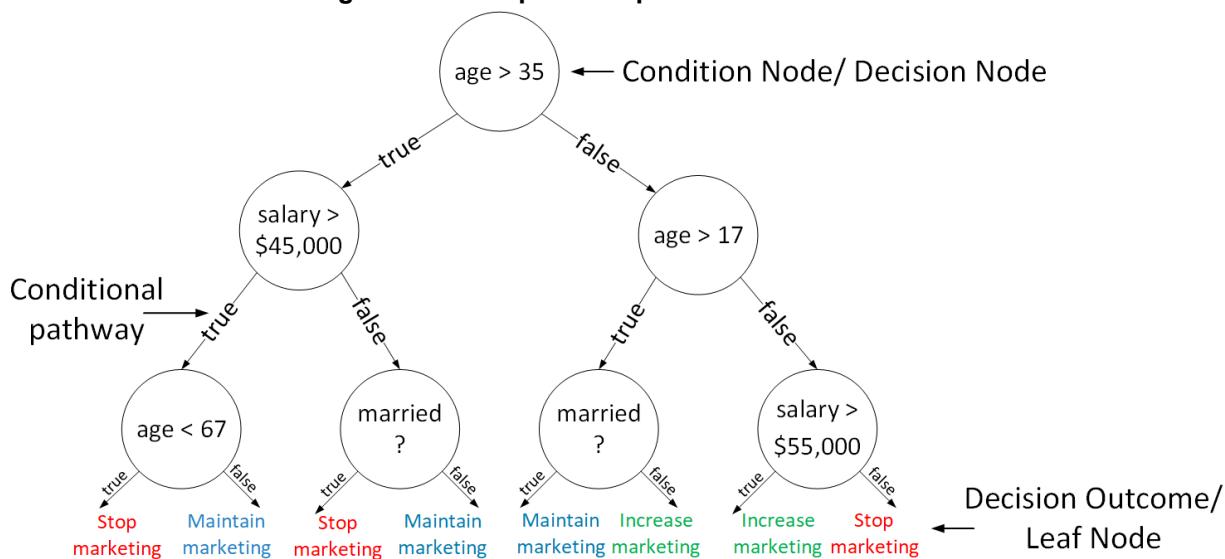
## Decision trees

Decision trees are a highly interpretable machine learning model. The patterns learned by many machine learning models are not easy to interpret. For example, neural networks consist of thousands to billions of parameters, such as weight and bias values, that map features to outcomes. Given the sheer number of parameters in these models, it is hard to understand why neural networks make the classifications and predictions that they do. Decision trees, however, are fairly simple from a conceptual standpoint. Further, the model learned by a decision tree algorithm can be easily visualized. The interpretability of decision trees can make them easier for managers to trust. However, they do not always provide the high levels of accuracy produced by neural networks and other advanced machine learning models.

Decision trees are so named because they are a tree data structure consisting of many branches of binary decisions. In this way, they are like the binary trees discussed in the chapter about data structures. Decision trees were briefly introduced in that chapter as well. Figure 11.7 shows the figure from the chapter on data structures. As you can see in the figure, decision trees consist of decision nodes and edges between those nodes. Each decision node within the tree data structure contains a condition that is either true or false. The decision nodes and connecting edges make decision trees a chain of if-else conditions that lead to a predicted outcome.

In the simple example below, the root node consists of a simple condition that checks if a customer's age is greater than 35. If the age is greater than 35, the decision tree checks to see if the customer's salary is greater than \$45,000. If false, it checks if the age is greater than 17. The tree continues branching to other decision nodes until a leaf node is reached. Leaf nodes contain the class that a data point belongs to (i.e., the predicted outcome). When the model is trained, each leaf node must contain data from only one class. A node with data from only one class is said to be **pure**. In the figure, this would mean that the various leaf nodes that end with the "Stop marketing" classification would only contain training data where the outcome labels were "Stop marketing".

**Figure 11.7: Simple Example of a Decision Tree**



To classify a new data point after training a decision tree, you would simply need to run the new data point through the decision nodes until you reach a leaf node and its predicted class. For example, assume you had a customer who was 40 years old, with a salary of \$75,000. Based on the tree in Figure 11.7, you would move to the left of the first decision node (i.e., 40 years old is greater than 35 years old), to the left of the second decision node (i.e., \$75,000 is greater than \$45,000), and to the left of the last decision node (i.e., 40 years old is less than 67 years old). This would put you at the leaf node with the class "Stop marketing." Based on this particular decision tree, we would stop marketing to the individual if we trusted the model. Although this was just described as a manual process, these actions would be accomplished with the sci-kit learn library using the predict() method after the model had been trained.

## Identifying splitting criteria

So, **how do you determine which criteria should exist at a decision node and how the conditional pathways will branch toward a specific leaf node?** The criteria at each decision node and the way that each level of the decision tree branches is determined during model training. As a reminder, training is the act of using a labeled data set to map feature patterns to labeled outcomes. In the case of decision trees, this mapping consists of selecting appropriate feature values to use as conditions at each decision node. The conditions at each decision node are called **splitting criteria**. In Figure 11.7, the splitting criterion for the root node is  $age > 35$ . During training, the decision tree algorithm determines the best criteria to split on at each node of the tree to create a reasonably efficient data structure. These splitting criteria are selected using a metric, such as information gain or the Gini index for classification problems, or variance calculations for regression problems.

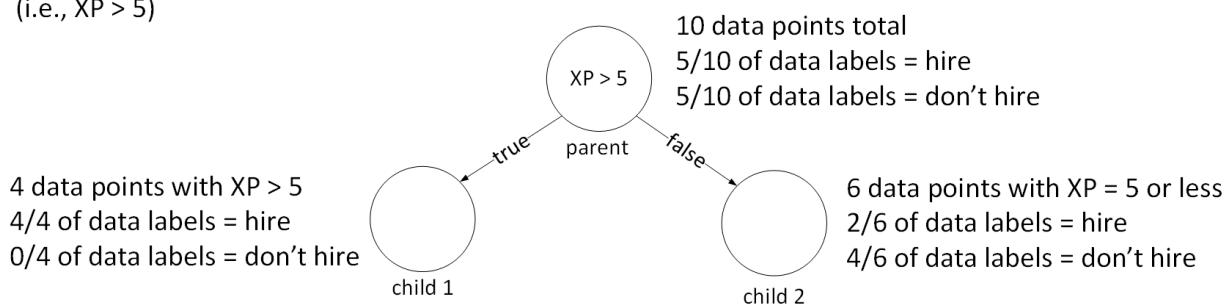
## Splitting based on information gain

**Information gain** is a measure of how well a particular splitting criterion reduces entropy within the data as the entropy of a node at one level of the tree is compared to the entropy of its child nodes on the next level. **Entropy** is a measure of how much impurity exists within the data of a particular node. As mentioned earlier, a pure node is one that contains data from only one class. The entropy for a pure node is 0. An impure node is one that contains data from multiple classes. The entropy for the most impure nodes is 1. The most impure nodes contain an even split of data across the classes.

For example, suppose you have two classes you want to predict: "hire" or "don't hire". A node with the highest level of impurity would contain 50 percent of the data as "hire" and 50 percent of the data as "don't hire". Figure 11.8 shows the example as a partial decision tree:

**Figure 11.8: Example Nodes for Information Gain Calculation**

Data set with 10 data points at the root.  
Assumes a split at the root node where  
work experience is greater than 5 years  
(i.e.,  $XP > 5$ )



The entropy of the data at a particular node can be calculated with the following **entropy equation**:

$$E(X) = - \sum p_i * \log_2(p_i)$$

**Where:**

- $E(X)$  is the entropy for the data set (i.e.,  $X$ ) at a given node
- $p_i$  is the probability of a data point belonging to class  $i$ . For example, assume you have 10 data points with 50 percent of the data belonging to class 1 and 50 percent of the data belonging to class 2:  $p_1$  would be  $5/10=0.5$  and  $p_2$  would be  $5/10=0.5$ .
- $\log_2(p_i)$  is the log base 2 of the probability of a data point belonging to class  $i$ .
- $-\Sigma$  is a mathematical symbol that tells you to perform a loop over the data points and subtract each of the values from one another. In the loop,  $p_i * \log_2(p_i)$  is calculated for each class and subtracted from the calculation of the previous class.

Building on the example in Figure 11.8, there are 10 data points at the root node with 5/10 data points belonging to the “hire” class and 5/10 data points belonging to the “don’t hire” class. With this data, the probability for “hire” would be 0.5 and the probability for “don’t hire” would be 0.5. Notice that the calculation of entropy of the example is 1. As stated earlier, the most impure nodes (e.g., 50/50 data split for a problem with two classes) have an entropy of 1 (i.e., max entropy). You can now see why the most impure nodes have an entropy of 1. Using the entropy equation, the calculations would look like this:

$$\begin{aligned} & - p_{\text{hire}} * \log_2(p_{\text{hire}}) - p_{\text{don't hire}} * \log_2(p_{\text{don't hire}}) = \\ & - \frac{5}{10} * \log_2(\frac{5}{10}) - \frac{5}{10} * \log_2(\frac{5}{10}) = \\ & - 0.5 * -1 - 0.5 * -1 = \\ & 0.5 + 0.5 = 1 \end{aligned}$$

To find the information gain, the entropy must be calculated for a node and for each of its child nodes. The information gain equation is simply the entropy value of the parent node subtracted from a weighted average of the entropy of its child nodes.

The **information gain equation** looks like this:

$$IG = E(\text{parent}) - (\sum w_i * E(\text{child}_i))$$

**Where:**

- $IG$  is the information gain
- $E(\text{parent})$  is the entropy calculation of the parent node
- $w_i$  is the number of data points in  $\text{child}_i$  of the parent node divided by the total number of data points in the parent node. In Figure 11.8,  $w$  of  $\text{child}_1$  would be  $4/10$  and  $w$  of  $\text{child}_2$  would be  $6/10$ .
- $E(\text{child}_i)$  is the entropy of the  $i^{\text{th}}$  child node.
- The  $\Sigma$  is a mathematical symbol that tells you to perform a loop over each child node and add together the weighted entropy scores for each child. This produces the weighted average entropy of the child nodes.

Previously, we calculated that for the example in Figure 11.8,  $E(\text{parent}) = 1$ . If you were to run the entropy calculation on each of the children in the figure, you would find that  $E(\text{child } 1) = 0 + 0 = 0$  (i.e., a pure node); and  $E(\text{child } 2) = 0.5283 + 0.3900 = 0.9183$ . The weighted scores of each child entropy score can now be calculated. The weighted entropy for child 1 =  $0 * 4/10 = 0$ . The weighted entropy for child 2 =  $0.9183 * 6/10 = 0.5510$ . The two weighted entropy scores can be added together to get the weighted

average entropy:  $0 + 0.5510 = 0.5510$ . Last, the entropy of the parent can be subtracted from the average weighted entropy of the children:  $1 - 0.5510 = 0.4490$ . The information gain for the example in Figure 11.8 would be 0.4490.

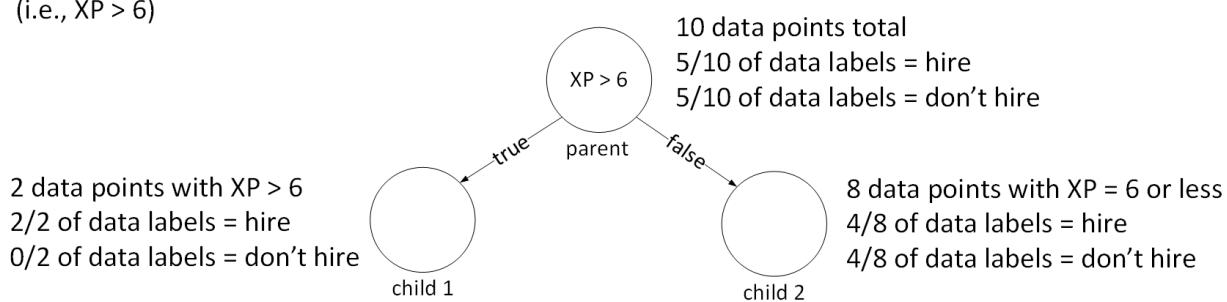
Now that you have an idea of what information gain is, you must now recognize that there are many possible splitting criteria that could be selected. In Figure 11.8, the example is based on a splitting criteria of  $XP > 5$ . You could have also calculated the information gain for  $XP > 6$ ,  $XP > 3$ ,  $XP < 4$ , etc. You could have also calculated the information gain of splitting on a different feature (e.g., age or salary). The magic behind the decision tree algorithm is in the selection of which criterion is used at each node. The decision tree algorithm seeks to maximize the information gain at each node by exploring the information gain of the possible splits that could be made.

For example, the decision tree algorithm in the running example would also calculate the information gain for  $XP > 6$ . Figure 11.9 shows what this split might look like. The information gain for this new criterion would be  $1 - (0 + 0.8) = 0.2$ . Based on the information gain scores, the  $XP > 5$  split (i.e., information gain = 0.4490) would be better than the  $XP > 6$  split. Assuming that  $XP > 5$  had the highest overall information gain score,  $XP > 5$  would be selected as the splitting criteria for the node. When you begin running decision trees with Python, you will notice that training a decision tree takes more time than you are used to waiting for a program to run. During this longer processing time, the algorithm is calculating information gain for different possible splits at each node. Now you can respect why the algorithm takes time to train.

**Figure 11.9: Example Nodes for Information Gain Calculation with Different Splitting Criterion**

Data set with 10 data points at the root.

Assumes a split at the root node where work experience is greater than 6 years (i.e.,  $XP > 6$ )



The following video provides a nice visualization of the process of choosing splitting criteria using the information gain metric: <https://www.youtube.com/watch?v=ZVR2Way4nwQ>.

### **Splitting based on Gini impurity**

Gini impurity is another metric that can be used to select splitting criteria for decision trees. The Gini impurity calculations tend to be faster and simpler than the information gain calculations. As such, the Gini impurity metric is the default in Python's sci-kit learn library for decision trees. Like entropy and information gain, Gini impurity scores range between 0 and 1.

A **Gini impurity score of 0 means that all of the data points in a node are of the same class** (i.e., a pure node). A Gini impurity score of 0.5 means the data is equally spread across all of the classes. A **Gini**

**impurity score of 1 means that the data points are equally likely to belong to any class.** Gini impurity is calculated as 1 minus the Gini score of the data at a decision node. The Gini score is a measure of purity. Thus, 1 minus the Gini score is a calculation of impurity.

The **Gini impurity equation** is shown below. As you can see, it is far simpler than the information gain calculation, which requires the use of the log function.

$$GI(X) = 1 - \sum(p_i)^2$$

**Where:**

- GI is the Gini impurity score for the data X
- $p_i$  is the probability of a data point belonging to class i. For example, assume you have 10 data points with 50 percent of the data belonging to class 1 and 50 percent of the data belonging to class 2:  $p_1$  would be  $5/10=0.5$  and  $p_2$  would be  $5/10=0.5$ .
- The  $\Sigma$  is a mathematical symbol that tells you to perform a loop over each child node and add together the probability of each class squared.

For example, in Figure 11.8, the Gini impurity score for **child 1** would be 0 based on the calculations shown below (i.e., a pure node):

$$\begin{aligned} 1 - (p_{\text{hire}}^2 + p_{\text{don't hire}}^2) &= \\ 1 - (\frac{4}{4}^2 + \frac{0}{4}^2) &= \\ 1 - (1 + 0) &= \mathbf{0} \end{aligned}$$

The Gini impurity score for **child 2** would be 0.4444 based on the calculations shown below:

$$\begin{aligned} 1 - (\frac{2}{6}^2 + \frac{4}{6}^2) &= \\ 1 - (0.1111 + 0.4444) &= \mathbf{0.4444} \end{aligned}$$

In order to achieve a Gini impurity score of 1, the probability of data belonging to each class would need to be 0 as shown in the calculations below.

$$\begin{aligned} 1 - (\frac{0}{6}^2 + \frac{0}{6}^2) &= \\ 1 - (0 + 0) &= \mathbf{1} \end{aligned}$$

With a labeled data set, you will never get a Gini impurity score of 1 because the data labels are not uncertain. In training, impurity scores will range from 0 to 0.5. You saw an example above with a pure node and its value of 0 and the impossible value of 1. To achieve an impurity score of 0.5, the class probabilities would need to be equal as shown below.

$$1 - \left( \frac{3}{6}^2 + \frac{3}{6}^2 \right) =$$

$$1 - (0.25 + 0.25) = 0.5$$

Similar to the use of entropy in the last section, splitting criteria are selected not on the score of an individual node, but on the weighted average of child nodes. As such, the Gini impurity must be calculated for each child node and then weighted based on the percentage of parent node data belonging to a child node. Thus, the formula to calculate the **weighted average Gini impurity score** for a parent node's children would be:

$$WGI = (\sum w_i * GI(child_i))$$

**Where:**

- WGI is the weighted Gini impurity score for the child nodes of a parent node
- $w_i$  is the number of data points in  $child_i$  of the parent node divided by the total number of data points in the parent node. In Figure 11.8,  $w$  of  $child_1$  would be 4/10 and  $w$  of  $child_2$  would be 6/10.
- $GI(child_i)$  is the Gini impurity score given the data for node  $child_i$ .
- The  $\Sigma$  is a mathematical symbol that tells you to perform a loop over each child node and add together the weighted Gini impurity scores for each child. This produces the weighted average Gini impurity score of the child nodes.

For example, the weighted average Gini impurity score for the two child nodes in Figure 11.8 would simply require weighting the two Gini impurity scores calculated earlier. Child 1 had 4/10 of the data from the parent node and child 2 had 6/10 of the data. Using these values to weight the Gini impurity scores, you get a weighted average impurity score of  $0.4*0 + 0.6*0.444 = 0 + 0.2666 = 0.2666$ .

Just like with information gain, you can calculate the Gini impurity for many different possible splits, such as  $XP > 6$ ,  $XP < 3$ ,  $XP = 4$ , etc.. Alternatively, you could choose to split on a completely different feature. Just like in the information gain example, the decision tree algorithm calculates the Gini impurity for the possible splits at any given node and then chooses the split with the lowest impurity score. With information gain, the goal is to maximize the gain. With Gini impurity, the goal is to minimize the impurity. Thus, the feature splitting criterion with the lowest impurity score is selected for a particular node. The weighted average impurity for the split in Figure 11.9 could also be calculated, resulting in a weighted score of  $(0.2*0 + 0.8*0.5) = (0 + 0.4) = 0.4000$ . The split in Figure 11.8 has a lower impurity score (i.e., 0.2666) and would therefore be selected over the splitting criterion in Figure 11.9.

Now that you have a basic understanding of the concepts and metrics underlying the decision tree algorithm, let's see how easy it is to train a decision tree in Python. With only a few lines of code, the sci-kit learn library will calculate all of the information described in this section.

### Implementing decision trees with Python

The sci-kit learn library contains a simple decision tree implementation. You simply need to import the following library into your notebook:

```
from sklearn import tree
```

Once the sci-kit learn tree library is imported you have access to both classification and regression trees. This chapter focuses primarily on classification trees. To create a new classification tree, you must instantiate the `DecisionTreeClassifier()` class.

```
classifier = tree.DecisionTreeClassifier()
```

Once you have instantiated a new classification tree, you can fit your data to the model using the `fit()` method you encountered in the section on k-means. The sci-kit learn library has a consistent API. You will find the `fit()`, `transform()`, and `fit_transform()` methods utilized across a multitude of sci-kit learn machine learning models. You will need to provide the `fit()` method with at least two arguments: the data features and the data labels.

To `fit()` the model, the features need to be separated from the labels into two distinct data structures. In this chapter, we will assume your data is contained within pandas data frames. To split the features from the labels, you can use **subsetting** methods to create a data frame with your features and another with just your labels.

For example, assume you had stock data that contained simple moving averages for different time intervals (i.e., 5-days, 10-days, 20-days, 50-days, 100-days, and 200-days) for each day the stock market was open and whether the price of the stock went up or down from the previous day's price. Suppose you wanted to use the simple moving average features (i.e., SMA5, SMA10, SMA20, SMA50, SMA100, and SMA200) to predict whether the closing price went up or down. To separate the features (i.e., X) from the entire data frame, you would do the following subsetting in pandas:

```
X = data[["SMA5","SMA10","SMA20","SMA50","SMA100","SMA200"]]
```

To separate the labels from the entire data frame (i.e., y), you would simply set your label data frame equal to the column containing your labels. In this case, assume the name of the column containing the labels is `priceChange`.

```
y = data.priceChange
```

Once you have separated your X variables (i.e., features) from your y variables (i.e., labels), you can separate your X and y data sets into data to train the decision tree and data to test the validity of the trained model. One of the simplest ways to split your data into training and test data sets is with the sci-kit learn library. The sci-kit learn library has a module to split data sets. The following import statement will give you access to a method that will convert your X and y data into X and y training and test data sets.

```
from sklearn.model_selection import train_test_split
```

Once imported, you can pass your separated X and y data frames into the `train_test_split()` method and it will output a list of data sets that can be used for training and testing a model. The order of the data set outputs is: XTrain, XTest, yTrain, and yTest. For example:

```
XTrain, XTest, yTrain, yTest = train_test_split(X, y)
```

To implement a decision tree that you can train and test with this data, start by importing the tree module within the sci-kit learn library.

```
from sklearn import tree
```

Once imported, you can create a new decision tree classifier by calling the `DecisionTreeClassifier()` method of the tree module, like this:

```
classifier = tree.DecisionTreeClassifier()
```

To train the decision tree using your training data set, you would simply call the `fit()` method on the decision tree classifier that you created. You will pass the fit method the feature training data and the label training data, like this:

```
classifier.fit(XTrain,yTrain)
```

To test the model, you can simply call the `score()` method built into the library. This time, you will pass it the feature test data and the label test data, like this:

```
classifier.score(XTest,yTest)
```

With the eight lines of code above, you can prepare a data set, split your data into training and test sets and train and test a decision tree classifier model. You can learn about the advanced options (e.g., changing selection criteria, choosing the max depth of branches for a tree, etc.) for training decision tree classifiers with the sci-kit learn library here:

<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

## **Random forests**

Random forests are a type of machine learning model known as an ensemble method. **Ensemble methods** consist of multiple machine learning methods combined together into a larger model. In the case of a random forest, the ensemble is simply the combination of multiple decision tree models. Decision trees are prone to overfitting the model to the training data set. This means that a simple change to the data set can create noticeably different accuracy results. To minimize this concern, random forests perform the decision tree algorithm multiple times on random subsets of the larger data set. By combining the results of these multiple decision trees, an “average” result can be obtained to provide a more consistent prediction.

For classification problems, random forests use a “voting” method to determine the class of a data point. A voting method counts the number of times a particular class was predicted by the individual decision trees that make up the random forest. The class with the highest count (i.e., highest votes) is the class that is predicted by the random forest. For regression trees, the mean of the values output by the individual decision trees is calculated. That mean value is what the random forest returns to the user.

## **Implementing random forests with Python**

Similar to decision trees, random forests are simple to implement in Python. First, you will want to separate your data set into X (i.e., feature) and y (i.e., label) data sets. Then, you will need to split your X and y data sets into XTrain, XTest, yTrain, and yTest data sets for training and testing. This can be accomplished with the same method shown in the previous section. To create a random forest, you will need to import the ensemble module of the sci-kit learn library:

```
from sklearn import ensemble
```

Once the ensemble module is imported, you can create a random forest classifier by calling the RandomForestClassifier() method.

```
ensembleClassifier = ensemble.RandomForestClassifier()
```

Following the same format as the decision tree classifier, you can then train the random forest using the fit() method and test the trained model using the score() method.

```
ensembleClassifier.fit(XTrain,yTrain)
```

```
ensembleClassifier.score(XTest,yTest)
```

As before, you can perform all of the calculations of multiple combined decision trees (i.e., a random forest) with only a few lines of code. You can learn more about the RandomForestClassifier() method and advanced features here:

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

## **Confusion matrices**

In the decision tree and random forest examples, you saw a model fitted to training data and then validated with test data. The score() method of the decision tree and random forest classifier simply provides the fraction of data points from the test data set that were accurately predicted by the algorithm. For example, the output 0.65 would mean that the model successfully predicted 65 percent of the labels for the test data set. Unfortunately, this simple metric isn't always enough to make good business decisions.

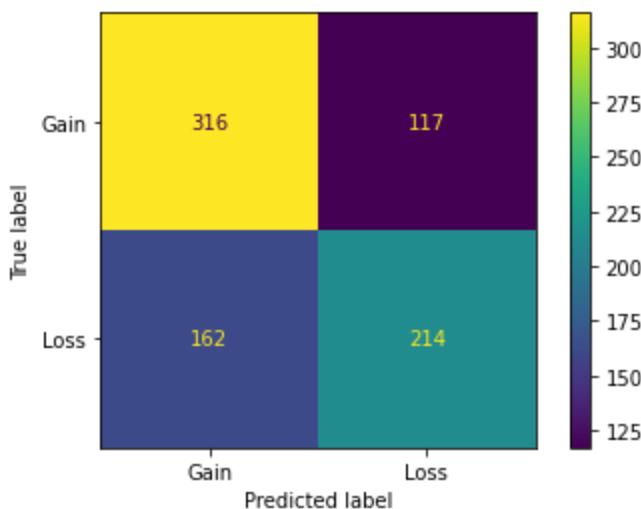
Suppose your current method of prediction is only 60 percent accurate. Should you automatically choose the newly trained model because it has a higher accuracy score? It depends. You need to understand where the prediction errors are generated and what type of errors are most harmful for your use case. Successful predictions are classified as either true positives or true negatives. For example, Assume you were predicting a gain or loss in the stock market as your classes. A **true positive** would be a prediction of a gain that actually resulted in a gain. A **true negative** would be a prediction of a loss that actually resulted in a loss. Just as there are two ways to classify successful predictions, there are two ways to classify incorrect predictions. A **false positive** is a prediction that was predicted to be positive (i.e., a gain in the market), but in reality was negative (i.e., a loss in the market). A **false negative** is a prediction that

was predicted to be negative (i.e., a loss in the market), but in reality was positive (i.e., a gain in the market).

By understanding where your data was successful (i.e., true positives and true negatives) and where mistakes were made (i.e., false positives and false negatives), you can make better decisions about the usefulness of the predictions. For some problems, false positives are far more harmful than false negatives. For other problems, false negatives are more harmful than false positives. Subject matter experts can help you determine which type of error is more harmful for a particular use case. Similarly, the accuracy of some models may appear better than it is when your data labels are not balanced (i.e., you don't have the same number of positive and negative classes in the data set). If your data set consists of mostly negative classes and your model is excellent at predicting the negative labels, your model may have a high accuracy score even if its ability to predict positive cases is poor. If correctly classifying positive cases were more important than correctly classifying negative cases, you would not want to use such a model.

To help you determine where errors are occurring in your data, you can utilize a confusion matrix. Figure 11.10 shows an example of a confusion matrix for the example of stock gains (i.e., the positive class) and stock loss (i.e., the negative class). Confusion matrices show the actual labels from the test data set along the rows of the matrix and the predicted labels along the columns. Confusion matrices show **true positives** (i.e., the top left corner), **true negatives** (i.e., the bottom right corner), **false positives** (i.e., the bottom left corner), and **false negatives** (i.e., the top right corner). The values within each cell of the confusion matrix represent the frequency of data points classified. In the example below, 316 data points were correctly labeled as gains and 214 were correctly labeled as losses.

**Figure 11.10: A Confusion Matrix of Stock Gains and Losses**



You can create confusion matrices in Python with the sci-kit learn library. To begin, you must import the metrics module. From the metrics module, you will import the `confusion_matrix()` method and the `ConfusionMatrixDisplay()` method. The `confusion_matrix()` method will calculate the data for the confusion matrix and the `ConfusionMatrixDisplay()` method will format the confusion matrix to be more readable.

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Once the confusion matrix modules are imported, you can use them. To do this, you will need the yTest data set and the y predictions from the trained model. Label predictions can be obtained by calling the predict() method on a trained model and passing it the XTest data set.

```
yPred = ensembleClassifier.predict(XTest)
```

```
confusion_matrix(yTest, yPred)
```

To make the confusion\_matrix() more readable, you will pass the confusion\_matrix() as an argument to the ConfusionMatrixDisplay() method, like this:

```
ConfusionMatrixDisplay(confusion_matrix(yTest, yPred), display_labels =  
ensembleClassifier.classes_).plot()
```

In the example above, the confusion\_matrix() is the first parameter. The second parameter extracts the distinct classes that exist in the classifier model by calling the classes\_ attribute on the classifier. These classes are arguments for the display\_labels parameter of the ConfusionMatrixDisplay() method. In Figure 11.10, the labels Gain and Loss were added to the confusion matrix through the display\_labels parameter. Last, notice that the plot() method is called on the ConfusionMatrixDisplay object. The plot() method will force the confusion matrix to be displayed in the cell output.

Don't accept a simple accuracy value when testing your predictive models. Use confusion matrices and similar tools and metrics to help you understand where the model is successful and where it makes mistakes. With the information provided in this chapter, you can now perform simple unsupervised and supervised machine learning analyses. This chapter is meant as a starting point for further exploration. Machine learning and forecasting is a broad discipline with many different models and algorithms to choose from.

## Chapter 11 Assignment

Please see the [Chapter Assignments Workbook](#) for the Chapter assignment.

# Chapter 12:

## Code Repositories

To this point, you have learned how to write simple programs and improve the user interface using only your personal computing device. Although writing code on a computing device is an important part of developing software, there is far more to the process than simply writing code. This chapter will introduce you to some of the tasks that the IT department must account for when developing software. Many tools and practices are utilized by the IT department to ensure that software is developed in a secure, high-quality, and consistent fashion. One movement that strives to improve software is the DevOps movement.

DevOps is a movement that seeks to better integrate the work of IT development teams and IT operations teams to produce better software more quickly. A **development team** uses programming knowledge to write code to build software. An **operations team** takes the code written by development teams and publishes it to production servers where users can gain access to the software. The operations team maintains the servers and other equipment that runs the software to ensure near constant software availability to users.

Although this process sounds simple, it is difficult to do well in practice. For example, developers' code may not work properly when put onto a production server because the developers didn't consider the environment on which the code would run. Similarly, an operations team might choose an inappropriate computing environment for the code written by a development team. To remedy issues like these, the DevOps movement seeks to integrate the workflows of the development and operations teams to minimize errors in moving code from development to production systems. Moving code from development environments to production environments is called **deployment**. Managing software deployment can be a stressful situation if you don't develop good deployment processes.

The DevOps movement is still developing and does not necessarily prescribe any particular practices. However, two important guiding principles of the DevOps movement include continuous integration and continuous deployment. **Continuous integration** is the act of maintaining a central repository of code so that multiple development teams can work from the same code base. A **repository** is a location for code to be stored and shared. Continuous integration ensures that all developers use the same code as they add new features to and fix bugs within software. Continuous integration is accomplished through the use of tools like git and platforms like GitHub and GitLab. Automated tools to build and test your code are another important part of continuous integration. Platforms like GitHub and GitLab contain tools to assist with the automated building and testing of software that your team writes. We will explore basic continuous integration technologies in this chapter, namely how to use code repositories.

**Continuous deployment** is a step beyond continuous integration that seeks to automate the process of moving code from development environments to production environments. Deployment has been handled manually in the past, because software needed to be manually tested before being deployed to production environments. Testing has long been a primarily manual process where quality assurance testers would scrutinize each feature and examine code for possible bugs and security issues. Although some organizations still rely solely on manual testing and deployment processes, most high-tech

organizations adopt automated or semi-automated continuous deployment processes. Multiple technologies are used to facilitate continuous deployment.

We will not explore continuous integration or deployment in depth in this text. Notebooks are rarely deployed in the same fashion as other types of software, such as web, mobile, or desktop applications. Notebooks tend to be used for presentations, data exploration, and the development of analytical models. These models are later integrated into traditional software systems that rely on continuous integration and deployment tools and practices. Still, you should at least know what a code repository is and how to use them. Code repositories are an important part of the business of writing code.

## Code Repositories with Git

Git is a standard and widely used tool that helps you build code repositories, place code within repositories, retrieve the latest version of code from repositories, manage versions of your code, and more. To begin, you need to make sure that git is installed on your computer. You can download git for Windows at: <https://git-scm.com/download/win>. To make things simple, follow the recommended settings. You can download git for Mac at: <https://git-scm.com/download/mac>. If you are using Linux, git comes preinstalled with several Linux distributions.

### Git and the command line

Once you have installed git, you can begin to use it. Although some GUIs exist to help you work with git, git was originally designed as a command line tool. Even if you do not become an expert at the command line, it is beneficial for you to know a little about how the command line works and how to perform basic functions in the command line. Many computing programs, particularly those used by IT department employees, are not written with GUIs. Instead, programs are executed through the command line.

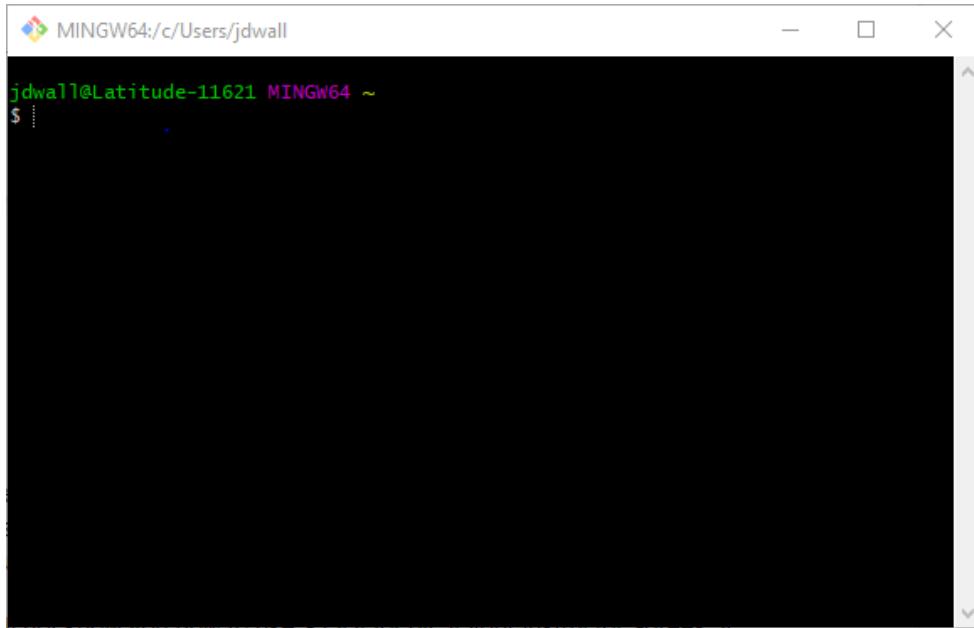
If you installed git for Windows from the above source, you should be able to run git either from the Command Prompt or from the **Git Bash command line tool**. The Git Bash command line tool will allow you to write both Windows and Linux style commands and will be used for the examples in this text. If you are running a Mac or Linux operating system, you can simply use your terminal instead of Git Bash.

There are some **basic command line prompts** that you should learn before diving into git commands. First, let's familiarize you with what the command line looks like. Figure 12.1 shows a command line. The line with the dollar sign is the location where you will type commands. After typing a command, press Enter on the keyboard to execute the command. After executing a command, the command line will either provide an error message, the output of the executed command, or nothing. In the command line, no feedback is often a sign that a program or command was executed properly. So don't be surprised if many commands don't show anything after you execute the command. In each case, a new command line will typically be provided after you execute a command so you can type your next command.

Some commands are executed based on where you are located within the filesystem of your computer. By default, the Git Bash command line prompt launches within the operating system's "user" folder for the logged in user. To find out which filesystem folder the command line is operating from, simply type **pwd** (for Linux/Unix or Git Bash) or **cd** (for Windows Command Prompt). The output will show the name of the folder you are currently in. The cd command doesn't work within the Git Bash tool even if you are working

within Windows. When using Git Bash, **type the pwd command** to find out which filesystem folder you are working within.

**Figure 12.1: Git Bash Command Line Tool**



The following are some common and basic command line prompts that you should familiarize yourself with. In general, the Git Bash tool works best with Linux commands. Don't worry about the Windows commands unless you are running git from the Windows Command Prompt. Many other basic commands exist beyond those described below.

Windows command	Linux command	Purpose
cd	pwd	The cd and pwd commands show you which file folder (i.e., directory) you are working from (i.e., the working directory). The cd keyword stands for current directory and pwd stands for path to working directory.
dir	ls	The dir and ls commands allow you to view the contents of the folder (i.e., directory) you are in. Dir stands for directory and ls stands for list.
cd [directory path]	cd [directory path]	The cd command, which stands for change directory, allows you to change the folder (i.e., directory) you are in. To use the cd command, type cd followed by the path to the directory you wish to move into.  For example, the following line will take you to the directory representing the Windows Desktop if the command line is already within the /c/Users/myuser directory. This is called a relative path because it moves to the requested folder relative to your current

		<p>folder location:</p> <p><b>cd Desktop/</b></p> <p>Alternatively, you can provide the full path to the directory. Building on the last example, if you weren't already within your "user" directory, you could cd into the Desktop folder by providing the full path to the Desktop:</p> <p><b>cd /c/Users/myuser/Desktop</b></p> <p>To go into a parent folder, you can type the following command. For example, if you were in /c/Users/myuser/Desktop and ran the command below, it would move you into /c/Users/myuser.</p> <p><b>cd ..</b></p>
--	--	---

With an understanding of how to navigate to your projects, you are now ready to create a repository for your code. To begin, you would use the cd and ls commands to move through the filesystem to get to the destination folder where you project files are stored.

## ***Creating/initializing a git repository***

Creating a git repository is fairly simple. Use the pwd, ls, and cd commands to help you move to the root folder of your project. If you are in the project folder (i.e., the folder where your Python code is located), you simply need to type the following command in the command line and press Enter.

**git init**

Alternatively, if you are not in the project folder, you can type git init followed by the full path to the project folder and press Enter, such as:

**git init /c/Users/myuser/myproject**

Make sure that you are in the correct directory or that you provide a full path to the project's root directory before initializing a new repository. Until you become comfortable with the command line, use the pwd command to ensure that you are in the correct directory.

## ***Checking the status of a repository***

The **git status** command allows you to keep track of what has happened to the files within your repository folder. Files like your Python notebook files can be untracked; tracked, but not committed; or committed. **Untracked files** are those that have not been added to the newly created repository, but exist within the project folder. **Tracked files** are those that have been added to the repository in a staging area, but not committed to the repository. The next section describes how to add a file to be tracked by the repository. **Committed files** have been added and committed to the repository to create a new version of the

repository. To check the status of files within your repository, type the status command into the command line and press Enter:

```
git status
```

## ***Adding files and folders to a repository***

Adding a file to a repository only places the file within a staging area to be committed later. Files added to a repository, but not committed are not a part of the current version of the repository. To add a file to the staging area, type the add command followed by the name of the file and press Enter. For example:

```
git add myNotebook.ipynb
```

You can also add an entire directory within your repository's root directory by using the name of the directory. For example:

```
git add folderName
```

Finally, you can add all untracked files and folders within the repository's root directory to the staging area by using one of the following commands:

```
git add .
```

```
git add -A
```

For the add command to work correctly, make sure that you are within the repository's root directory by using the cd command.

## ***Committing changes to a repository***

After adding files and folders to a repository's staging area, you can commit them with the commit command. You must add files to the repository using the add command before committing them. The commit command creates a new version of the repository with the newly added or altered files and folders. Creating new versions of the repository allows you to revert back to a previous version if you later find an error in the code. Commits usually include a message that allows other developers to know what changes were made in the commit (e.g., adding a new feature, fixing a bug in a file, etc.). The -m option allows you to add a message to a commit. It is best practice to include a commit message with every commit. For example:

```
git commit -m "added feature to calculate compounded interest"
```

Depending on how you installed git, you may need to configure the git user on your operating system before committing. If you receive an error stating that you need to specify a username and email, run the commands below. To configure git, you will use the git config command. If needed, set up the user.name and user.email properties. You will run each command separately by pressing Enter after typing each command. For example:

```
git config --global user.email "email@email.email"  
git config --global user.name "jeff"
```

The user information is an important part of the repository. It allows the repository to keep track of who is committing changes to the code base.

## ***Local versus remote repositories***

To this point, you have used git to work with local repositories. A **local repository** is a repository that exists only on your personal computing device. Local repositories are useful for developers as they write code, but aren't accessible to the other members of a development team. Some software development projects require multiple team members and even multiple teams, so sharing code in an efficient manner is crucial. To share code between developers or across development teams, a remote repository is needed. A **remote repository** is a repository that resides on a server and is accessible via a network, such as the Internet. Remote repositories are often hosted on platforms such as GitHub and GitLab.

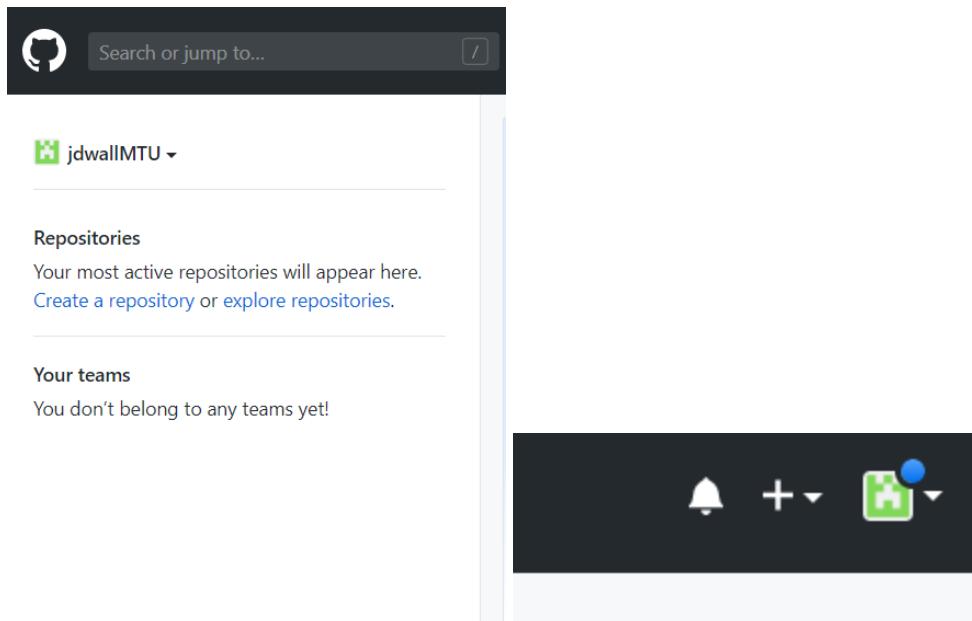
## **Continuous Integration through Git and GitHub**

Now that you understand the basics of repositories and git, you can begin to write and share code with team members through remote repositories. In this section, we demonstrate how to work with GitHub to create remote repositories and collaborate with teammates with a shared, remote repository. To complete the assignment in this chapter, you will need to sign up for a [GitHub account](#). GitHub offers many of its features for free. In this text, everything demonstrated can be accomplished with a free account. At the time of this writing, GitHub also offered free student accounts that provide access to advanced features.

### ***Creating a remote repository***

After signing up for a GitHub account, you can create remote repositories that can be shared with other developers. To create a new repository, click the “Create a repository” link in the left-hand panel of the GitHub home page (after logging in). Alternatively, you can click the menu icon in the top right hand corner of the page and click “Your repositories” from the menu list. On the repositories page, you will find a button labeled “New” that will take you to the page to create a new repository. Figure 12.2 shows the different links and icons to create a new remote repository on GitHub.

**Figure 12.2: Links to Create a Remote Repository on GitHub**



The repository creation page provides the form shown in Figure 12.3 to help you create a repository. Start by giving the repository a name. It is best not to include spaces in the name of your repository. You can use hyphens instead. Next, you can provide a description of what the repository will contain (i.e., what the software is). The description helps teammates or potential collaborators understand what project the repository is tracking. Unless you have signed up for a GitHub education account or have a paid GitHub account, you will need to make your repositories Public. Private repositories are available to paid customers of GitHub and may be available to those with a GitHub education account.

GitHub repositories can be created with some basic features. Most GitHub repositories include a ReadMe file that explains the software project and how to use the software. If you have certain files that you don't want to be included in the public GitHub repository (only on your local filesystem), you can add a .gitignore file. The .gitignore file tells the repository to ignore (i.e., not include) files, folders, or file types specified in the .gitignore file. You can also add a license file to explain how the software can be used by others who visit your remote repository. If your repository is public, anyone can download your code. The license explains the legal obligations of using the code. Once you have finished filling out the repository creation form, click "Create repository" and your remote repository will be created and become available.

**Figure 12.3: The Create a New Repository Form**

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?  
[Import a repository.](#)

Owner      Repository name \*

 jdwallMTU /

Great repository names are short and memorable. Need inspiration? How about [literate-couscous?](#)

Description (optional)

 Public  
Anyone can see this repository. You choose who can commit.

 Private  
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

Initialize this repository with a README  
This will let you immediately clone the repository to your computer.

Add .gitignore: None ▾ | Add a license: None ▾ 

**Create repository**

**Caution:** there are two ways to start a remote project:

1) you can create a local repository and then create a remote repository and push the code from the local repository to the remote repository. If you choose **option 1** to create a remote repository, it is best to **create your remote repository without ReadMe, .gitignore, or license files to avoid conflicts between the local and remote repository**. These files can be created after your initial push from the local repository to the remote repository.

2) you can create a remote repository and then clone the remote repository to create a local repository. If you choose option 2, you are safe to create the ReadMe and other files when creating the remote repository.

After creating the repository, you will be taken to a page that explains how to create a local repository on your computer that connects to the newly created remote repository. If you haven't already created a local repository, you will use the following code in the command line (running each line separately with the Enter key):

```
git init  
git commit -m "first commit"  
git remote add origin https://github.com/myusername/myprojectname.git  
git push -u origin master
```

Some of the code above (i.e., git init and git commit) should look familiar. Other commands are new. The **git remote add** command creates a tracking link between the local repository and the remote repository. The word “origin” is an alias that can be used to call to the remote repository instead of typing out the entire remote repository URL every time you want to do something with the remote repository. The URL is where your repository is located on GitHub. Last, the **git push** command takes the committed code in your local repository and puts it on GitHub. Recall that the origin keyword is used to represent the URL to your remote repository. So git push -u origin tells git to send the committed code on your local repository to the GitHub URL associated with the origin alias. The word master represents the branch that the code will be pushed to. Branches will be described in greater detail shortly.

Alternatively, you can just clone the newly created remote repository to your personal computing device. The **clone** command will create a copy of the remote repository on your local machine. The clone command will create a local repository with all of the code from the remote repository. You do not need to use the git init command if you choose to clone a remote repository. The clone command takes care of initialization for you. The clone command is passed the location of the remote repository, and can also be passed the name of the folder where the local copy of the repository should be placed. If no destination folder is provided, the clone command will create a folder with the name of the remote repository. The clone command is as follows with and without the destination folder (you would change the url with the url to your remote repository):

```
git clone https://github.com/myusername/myprojectname.git  
git clone https://github.com/myusername/myprojectname.git myprojectfolder
```

If you have already created a local repository before creating your remote repository (option 1 above), you can skip the git init and commit commands as you have executed these commands previously. To link your existing local repository with the remote GitHub repository, you would simply run the following code within the directory of your local git repository:

```
git remote add origin https://github.com/myusername/myprojectname.git  
git push -u origin master
```

## **Branches and workflows**

A **branch** is a version of the code within a repository. Every git repository is initialized with a default **master** branch (both local and remote repositories). Multiple branches of a repository can exist simultaneously, but only one branch can be active at a time. Branches can be created and deleted on command. Branches are created from an existing branch. When you create a new branch, the code history from the parent branch is copied to the new child branch. If you are in charge of creating a branch, you will create the branch on your local repository and then push it to the remote repository for others to use. If others have created a branch that you need to work on, you will pull the branch from the remote repository to your local repository.

Branches are often used by teams to create workflows. A **workflow** is the process used by a team to write code, share code, test code, and move code to a production environment. Some teams use branches to represent each major step in the workflow. For example, a team might create a permanent development branch, test branch, and production branch. The **development branch** would be used to move new code from each developer's local repository to a shared remote repository. The **test branch** would be used to track automated tests of your code. The **production branch** would contain code that had successfully passed all tests and was ready to be deployed to a production environment. This is only one of many possible workflows that can be created through GitHub and similar technologies.

Many teams view the **master branch** as an important branch that must be maintained with great care. For your own personal projects, the master branch may be the only permanent branch you use. Some teams use the master branch to represent the production branch in their workflow. Other teams use the master branch to represent the development branch. However, some teams don't use the master branch at all. These teams will delete or rename the master branch and use their own naming conventions for branches. Pay attention to the way your organization and team uses branches and follow their conventions. How you set up your workflow is important, but not as important as being consistent in the use of the team's workflow.

The “**development” branch is often the most widely used branch**. Whether the development branch is called development, dev, master, or something else, the repository branch dedicated to storing new features and bug/issue fixes should be maintained with care. Since many developers will share the same remote development branch, it is important that you avoid code conflicts. A **conflict** exists when the code on a local branch is different from the code on a remote branch or another local branch (e.g., the local branch has a class with an attribute named “foo” and the remote branch has the same class with an attribute named “bar”). Conflicts are caused by multiple developers pushing content to the same shared repository. Conflicts require developers to resolve any code conflicts before continuing with the workflow.

To avoid conflicts, teams often create **temporary branches from the development branch**. Branches of the development branch are often created for each new feature or bug fix. Each new feature or bug/issue is given a separate temporary branch. All code written on a developer's local machine is written within the feature or bug/issue branch. Once the feature or bug/issue code is complete, the temporary branch is merged into the development branch. **These temporary branches help to protect the development branch** from being overly cluttered with partially completed features or bug/issue fixes that could lead to many code conflicts. The development branch should always be ready to be pushed to the next branch (i.e., testing or production) and should not contain partially completed work.

The git branch command can be used to **view existing branches** and to create new branches. Only one branch is active at a time. To view the branches that currently exist within your repository, and which branch is the active branch, type the following into the command line and press Enter. The branch with an asterisk next to it is the active branch:

## git branch

**To create a new branch**, you will type the git branch command followed by the name of the new branch. The names of branches should NOT contain spaces. The newly created branch will be branched from whatever branch is the active branch. By default, this means that branches will be created from the master branch. For example, the following code would create a new branch called my-branch off of the master branch:

### **git branch my-branch**

When working within git, you can only work within one branch at a time (i.e., the active branch). To move from one branch to another, you will use the checkout command. By default, you will be working within the master branch. Based on the example above, assume you wanted to move from the master branch to the newly created my-branch. This would be accomplished by typing the following in the command line and pressing Enter:

### **git checkout my-branch**

To go back to the master branch, you could then type:

### **git checkout master**

If you make changes to the code on one of the branches and switch to another branch that doesn't have the change, you will notice in Eclipse or your other IDE that the code changes no longer exist. When you switch from branch to branch, you are actually loading the files and folders from that branch into the project directory. Make sure that you are working from the correct branch. Remember that you should not work directly in a permanent repository (i.e., master, development, test, production, etc.). Rather, you should create a new temporary branch for the specific feature or issue you are working on. Then, you can merge the temporary branch with the development branch when you are finished. In a later section, you will learn how to merge branches. When you create a branch on your local repository, you will want to push the entire branch to your remote repository as a backup of your code and to allow other developers who might be working on the same feature to share the code. To push a branch to your remote GitHub repository, you will type the following command, making sure to change [your branch name] to the name of your branch, and press Enter:

### **git push -u origin [your branch name]**

For example:

### **git push -u origin my-branch**

If you visit GitHub after typing this command, you will notice that your remote repository now has multiple branches.

## ***Pushing to and pulling from a remote repository***

In the previous sections, you learned how to create a remote repository on GitHub and push content from a local repository to the remote repository or how to clone a remote repository as a local repository. The ***push command can be used any time you need to move committed changes from your local repository to a remote repository.*** Developers traditionally wrote complex features within their local repositories and only pushed changes to the remote repository after weeks or even months of work. Unfortunately, this practice can lead to complex conflicts between the code in a developer's local repository and the code in the shared, remote repository.

**Continuous integration** seeks to minimize such conflicts by encouraging developers to regularly test and push their local code to shared, remote repositories. Regular pushes help to minimize the complexity of conflicts, because developers push smaller amounts of code, which leaves less opportunity for conflict. Although conflicts may still arise, they are likely to be simpler to resolve. Organizations have different definitions for what “regular” means. Such definitions can go from extremes of: pushing to the remote repository every time you write a new method to pushing content after completing a feature, which could be a week or more. Other organizations may have time-based rules for pushing content to the remote repository, such as at the end of each day or once per hour. Recall that most teams prefer to push to temporary branches and then merge the temporary branches into permanent branches.

In addition to pushing content regularly, ***it is advisable to pull the most current content from the remote repository regularly***. Pulling can be particularly important just before you push code to the remote repository. **Pulling** content from a remote repository means updating your local repository with code from the remote repository that may have been changed by other members of your team. For example, teams might have a practice of pulling content from the remote repository at the beginning of each day and every hour throughout the day. The pull command is simply:

### **git pull**

You can specify the branch as well, which is important when dealing with multiple branches. For example:

### **git pull origin master**

When you pull content, you may experience some conflicts between what you have written on your local repository and the changes made by your teammates; meaning that your code may not work as expected because someone removed a class that you needed or changed the attributes or methods of a class. You will need to resolve such conflicts before pushing your code to the remote repository. Conflicts often require communication between team members. Some teams use tools like Slack to communicate updates being merged to important branches or to identify changes to important classes within the repository. Ideally, you should never change the structure of a class (i.e., its attributes and methods) once it has been created to avoid breaking other code that relies on the class. However, it can be necessary at times to reconfigure the code to optimize it or make it easier to use.

## ***Pull requests and merging branches***

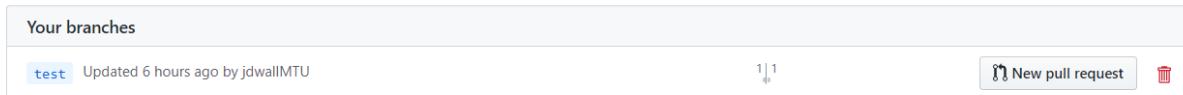
As you create temporary feature and bug/issue branches, you will eventually need to merge them into the permanent remote branches that make up your team’s permanent repository. Although there are multiple ways to accomplish this action, we will discuss one of the safer approaches. This section will show you how to merge your local temporary repository into a remote permanent repository. Unless your team tells you otherwise, never push content directly to a permanent remote repository (e.g., master, development, test, production); always push to a temporary remote repository and then merge the temporary repository with the appropriate permanent repository.

Once you have finished creating a new feature on your local feature repository, you will want to do the following to merge the feature branch into the remote permanent branch (e.g., master or development):

1. Make sure you have added and committed all changed files on your local feature repository
  - git add .

- git commit -m "commit message goes here"
- 2. Pull updated content from the temporary remote repository if multiple developers are working on the same feature branch. This will update your local feature repository with any changes made by other developers.
  - git pull origin [name of remote feature repository]
- 3. Fix any conflicts that arise.
- 4. Push the updated local feature repository to the remote feature repository
  - git push -u origin [name of remote feature repository]
- 5. Visit the remote repository on github.com and navigate to the branches page
- 6. Find the remote feature repository branch under the “Your branches” section and click the “New pull request” button as depicted in Figure 12.4.
- 7. Continue as described below

**Figure 12.4: Creating a Pull Request on GitHub.com**



After clicking the “New pull request” button, you will be taken to the form shown in Figure 12.5. The pull request form will either state that you are able to merge, or that you can’t automatically merge. Either way, you can use the form to create a pull request. Once you fill out the form, press the “Create pull request” button. If you are not able to merge the branches, you will be given the opportunity to resolve any code conflicts after creating the request.

**Figure 12.5: Pull Request Form on GitHub.com**

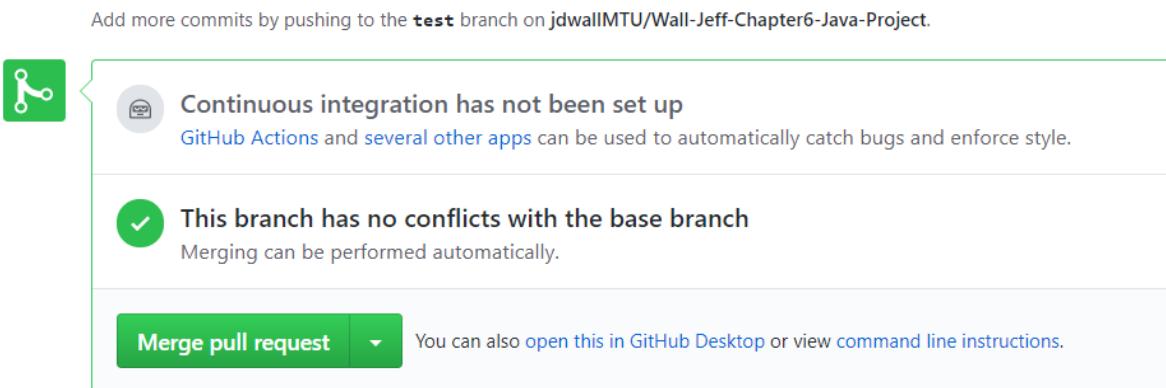
## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

 A screenshot of the GitHub 'Open a pull request' form. At the top, it shows the base branch as 'master' and the compare branch as 'test'. To the right of these dropdowns, a green checkmark icon indicates 'Able to merge. These branches can be automatically merged.' Below this, there is a text input field containing 'final comment' and a rich text editor toolbar. A large text area labeled 'Leave a comment' is present. At the bottom of the form, there is a file upload area with the placeholder 'Attach files by dragging & dropping, selecting or pasting them.' and a large green 'Create pull request' button. To the right of the form, there are several configuration sections: 'Reviewers' (No reviews), 'Assignees' (No one—assign yourself), 'Labels' (None yet), 'Projects' (None yet), and 'Milestone' (No milestone).

After clicking the “Create pull request” button, if the merge had no conflicts, you will be asked whether you want to merge the pull request. Click the “Merge pull request” button depicted in Figure 12.6 and then provide a comment for the merge (like the -m comment on a git commit command). Last, click the “Confirm merge” button.

**Figure 12.6: Merge the Pull Request on GitHub.com**



Once you have merged the pull request, the permanent branch that you merged to will have the code from the feature branch. At this point, the feature branch can be deleted. After confirming the pull request, you will be given an option to delete the temporary feature branch. Since the feature is complete, you can delete the branch on the remote repository. You will still need to delete the branch from your local repository. Deleting a branch from a local repository can be done with the git branch command by including the -d option and the name of the branch you wish to delete. If the branch you wish to delete is the active branch, you will first need to checkout another branch, such as the master branch. For example:

```
git checkout master  
git branch -d feature-branch-name
```

You will also want to pull the updated master/development branch from your remote repository to your local repository so that your local repository is up to date (ex. git pull origin master).

If you run into a code conflict, you will see the message depicted in Figure 12.7. You can still create the pull request like before.

**Figure 12.7: A Pull Request Form with a Merge Conflict on GitHub.com**

## Open a pull request

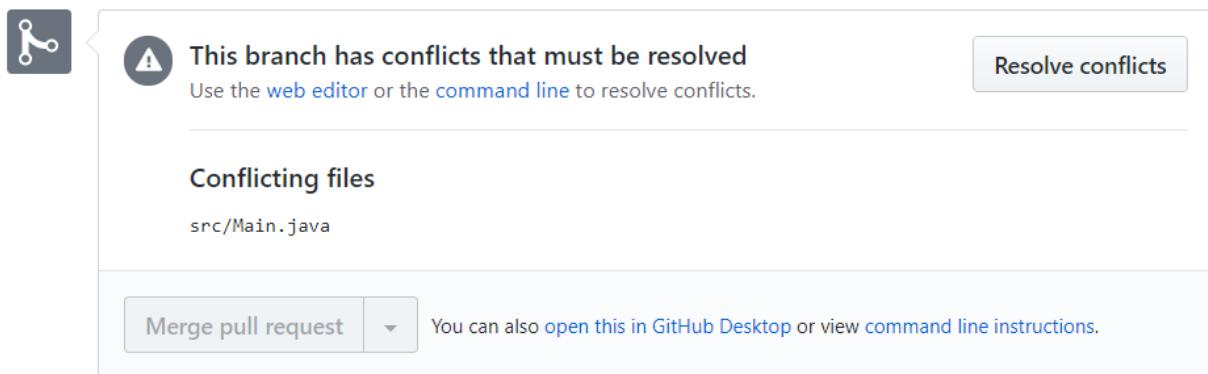
Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

The screenshot shows the GitHub pull request creation interface. At the top, it says "base: master" and "compare: test". A red warning message "Can't automatically merge" is displayed. The main area has a title "the correct comment" and two tabs: "Write" (selected) and "Preview". Below the tabs is a text area with placeholder text "Leave a comment". To the right of the text area are various settings: "Reviewers" (No reviews), "Assignees" (No one—assign yourself), "Labels" (None yet), "Projects" (None yet), and "Milestone" (No milestone). At the bottom right is a green "Create pull request" button.

On the next page, you will click the “Resolve conflicts” button depicted in Figure 12.8.

**Figure 12.8: Resolve Merge Conflicts on GitHub.com**

Add more commits by pushing to the **test** branch on jdwalIMTU/Wall-Jeff-Chapter6-Java-Project.



After clicking the “Resolve conflicts” button, you will be taken to a text editor on GitHub.com showing you the conflicts between the code you pushed from the feature branch and the code on the permanent remote branch. Figure 12.9 shows the text editor for resolving conflicts. The text editor shows the conflicts with a yellow highlight on the code line numbers that are in conflict. The code between the <<<< and ===== symbols represents the code from your local repository that was not on the permanent remote branch. The code between the ===== and >>>> symbols represents code that was on the permanent remote branch, but not on the local repository.

To resolve the conflict, you must decide whether to delete the code you created on your branch, delete the code from the permanent repository, or retain both pieces of code. Regardless, you will need to delete the <<<<, =====, and >>>> symbols from the file. If you had other conflicts, you could use the Next and Prev buttons to solve other conflicts. When the “Mark as resolved” button becomes clickable, click it and then click the “Commit merge” button that pops up. This will take you back to a page similar to that shown previously for a push with no conflicts. Click the “Merge pull request” button after filling out the form and then confirm the merge. The code with the resolved conflicts will be merged into the permanent branch.

Again, you will want to delete the feature branch on the remote repository and on your local repository. You will also want to update your local permanent branches with the merge you made.

**Figure 12.9: Text Editor for Resolving Conflicts on GitHub.com**

The screenshot shows a GitHub text editor interface for a Java file named 'src/Main.java'. The code contains several comments and a conflict marker. Lines 18 through 22 are highlighted with a yellow background, indicating a conflict. The conflict markers are: '<<<< test' (blue), '//the correct comment' (green), '===== (grey)', '//a comment that needs to be deleted' (red), and '>>>> master' (orange). The status bar at the top right shows '1 conflict'. A 'Mark as resolved' button is visible in the top right corner of the editor area.

```
src/Main.java
1
2  public class Main {
3
4      public static void main(String[] args) {
5          SimpleProgram program = new SimpleProgram();
6          program.run();
7
8          System.out.println("We are on the test branch");
9
10         //this is another change to main
11
12         //applesauce is the best
13
14         //this is yet another comment
15
16         //this is a final comment
17
18     <<<< test
19     //the correct comment
20     =====
21     //a comment that needs to be deleted
22     >>>> master
23     }
24
25 }
26
```

After merging a feature, you can move on to another feature or bug/issue and go through the steps again. Continuing in this fashion will ensure that the permanent remote branches remain clean with completed features and bug/issue fixes. Git and remote repositories, such as GitHub and GitLab are useful tools for developers.

## Chapter 12 Assignment

Please see the [Chapter Assignments Workbook](#) for the Chapter assignment.

# Programming Syntax Cheat Sheet

This cheat sheet provides brief reminders of programming syntax so that you do not need to go back and dig through chapters for commonly used techniques in Python.

## Working with Variables

This section describes the syntax for declaring variables, initializing variables, and using variables. Variables are named storage buckets in computer memory that store data values, including primitive data types (i.e., integer, float, boolean, and string data) and entire objects.

### **Variable declarations**

To declare a variable means to create a named space in memory to store data. No data is included in the variable at the time of declaration. In Python, variables are declared by setting the variable equal to None (i.e., an empty variable):

```
variableName = None
```

### **Variable assignment**

To assign a variable a value means to place a data value into the variable after it has been declared. Variables can be assigned values multiple times throughout the execution of a program. Each time a variable is assigned with a new value, the old value is replaced. Each example below assigns the variableName variable with a new value, including with an integer, a float, a string, and an object.

```
variableName = 1  
variableName = 1.5  
variableName = "Jamie Doe"  
variableName = Customer()
```

**Caution:** Python is a dynamically typed programming language, which means that a variable is not limited to one data type. That is why variableName in the example above can contain an integer, float, string, or object. Although this provides flexibility, it can also cause errors since a variable can contain data types that don't make sense within your program (e.g., you can't multiply an integer by a string).

### **Variable initialization**

Variable initialization is simply the act completing variable declaration and assignment on one line. So, instead of:

```
variableName = None #variable declaration  
variableName = 1 #variable assignment
```

You would simply skip straight to:

```
variableName = 1 #variable initialization (i.e., declaration and assignment on one line)
```

## **Performing basic mathematical operations with variables**

As in algebra, variables can be used to contain data that is used in simple or complex calculations. The basic mathematical operations in Python include:

- Addition: + (e.g., six plus seven is:  $6 + 7$ )
- Subtraction: - (e.g., six minus seven is:  $6 - 7$ )
- Multiplication: \* (e.g., six times seven is:  $6 * 7$ )
- Division: / (e.g., six divided by seven is:  $6 / 7$ )
- Power of: \*\* (e.g., six raised to the power of seven is:  $6 ** 7$ )

Parentheses can be used to create calculations with more complex orders or operation. Variables can be included in calculations, the result of which is stored in another variable. Python follows the order of operations commonly taught in mathematics courses.

```
firstNumber = 6
secondNumber = 15.2
thirdNumber = 2

result = (firstNumber + 7) ** thirdNumber / (secondNumber * 14 - firstNumber) #would return 0.8172...
```

The value of the new variable result would be 0.8172... based on the numbers provided.

In mathematical notation, the example above would look like this:

$$\frac{(firstNumber+7)^{thirdNumber}}{(secondNumber \times 14 - thirdNumber)}$$

or with the variable values:

$$\frac{(6+7)^2}{(15.2 \times 14 - 6)}$$

## **Using variables within strings**

You will often want to use variables within strings to provide readable output to users. This can be accomplished in a variety of ways:

Assume first the creation of several variables that will be embedded into a string

```
name = "Jeff Wall"
age = 21
height = 69.5
```

String concatenation allows strings and variables to be “added” together with the plus sign. Notice that numeric variables need to be wrapped in the built-in str() method to convert them to strings. Only strings can be concatenated with this method.

```
result = name + " is " + str(age) + " years old and is " + str(height) + " inches tall."
```

The f-string method provides the easiest and most intuitive implementation. To use the f-string method, simply add the letter “f” to the front of a string before the opening quotation mark. Then, you can add variables directly into the string using curly brackets and the name of the variable.

```
result = f"{name} is {age} years old and is {height} inches tall."
```

The format() method is another way to embed variables into strings. Although easier than concatenation with the plus sign, the format() method is less intuitive than the f-string method. The format() method relies on ordered placeholders represented with curly brackets. Then, the format() method is called on the string object. The arguments passed to the format() method need to be in the order the placeholders appear in the string.

```
result = "{} is {} years old and is {} inches tall.".format(name, age, height)
```

In each case, the result variable above would contain: "Jeff Wall is 21 years old and is 69.5 inches tall"

## Working with Functions

This section describes how to create and use Python functions. Functions are like processes that take input values (i.e., parameters), perform logical operations on the input values to transform them into outputs (i.e., return values).

### *Creating a function*

Functions provide repeatable business logic that can be used over and over again with different input values (i.e., parameters). The basic syntax for a function is as follows:

```
def functionName(parameter1, parameter2,... parameterN):
    #body of the function is indented on the next lines.
    #The logic and the return statement for the function would go here
```

Remember that the keyword “def” is required before you enter the name of the function. Then a list of parameters is included within parentheses. Each parameters is simply a variable to store input values (i.e., arguments). The body of the function (i.e., the business logic) is indented after the function declaration. The return statement that returns the value after completing the business logic is also contained in the body of the function. For example:

```
def addTwoNumbers(firstNumber, secondNumber):
    #This is the body of the function where inputs are changed to outputs
    #The firstNumber is added to the secondNumber
    #The result of the summation is stored in a variable called "result"
    result = firstNumber + secondNumber

    #the return statement below returns the value stored in the "result" variable
    return result
```

## Using a function

Once a function is created, you can use it by calling the function and providing it with input values (i.e., arguments). In the example below, the `addTwoNumbers()` method in the previous section is called multiple times to calculate different values. To use a function, you simply call the name of the function and pass it arguments for each function parameter. The return value of a function can be stored in a variable for later use if desired. This is shown with the `functionResult` variable in the example below.

```
functionResult1 = addTwoNumbers(6, 7) #would store the value 13 in the variable functionResult  
functionResult2 = addTwoNumbers(2, 3) #would store the value 5 in the variable functionResult
```

In this example, the value 6 in the first example is stored in the `firstNumber` parameter of the `addTwoNumbers()` function and the value 7 is stored in the `secondNumber` parameter. Based on the `addTwoNumbers()` function logic, these numbers would be added to return the value 13.

The order of the values (i.e., 6 and 7 in the first example, and 2 and 3 in the second example) matter. The first value is entered into the first parameter, the second value in the second parameter, and so on. This is known as using ordered parameters.

Python has many pre-created functions built into the language itself. These functions can also be called in a similar fashion. You simply need to know the parameters and parameter order of these functions. For example, the built-in `print()` function will display the results of a variable to users. The `print` function accepts a value that will be printed to the console. Notice in the example below that the `print()` function is called. This time, the result of the function is not stored in a variable. That wouldn't make sense for the `print()` function.

```
print("Hello World")
```

## Calling functions with named parameters

Python also allows you to use named parameters when passing arguments into a function. Many languages don't provide this functionality. When using named parameters, you don't have to worry about the order of the parameters when the function was created; you simply need to know the names of the parameters. The examples below show the same two examples presented above, but with named parameters in a different order.

```
functionResult1 = addTwoNumbers(secondNumber=7, firstNumber=6) #would store the value 13  
functionResult2 = addTwoNumbers(secondNumber=3, firstNumber=2) #would store the value 5
```

## Working with Classes and Objects

Object-oriented programming relies on classes and objects to manage complex operations. Classes are representations of things that exist in the real world (e.g., customers, products, suppliers, assets, etc.).

Classes consist of two types of members: attributes/properties and methods. Attributes/properties are simply variables that store information about the class. For example, a `Customer` class might have the following attributes: `firstName`, `lastName`, `age`, and `phoneNumber`. Methods are actions that are performed by or on the class. Methods are simply functions that belong to a class. For example, a

Customer class might have methods (i.e., functions) to change the values of class attributes (e.g., changePhoneNumber()) or review purchase history (i.e., reviewPurchaseHistory()).

Classes are templates from which objects are created. An object is instantiated from a class. Objects contain actual data values, while classes simply possess the logic to describe how values are to be stored and manipulated. Objects are created from a class by calling the constructor method of the class. The constructor method of a class is always named `__init__()`. Two underscores before and after the word init. The constructor method often contains class attribute declarations.

## ***Creating a class***

The syntax for declaring a class consists of the keyword “class” followed by the name of the class (e.g., Customer, Product, etc.). Then, the body of the class (i.e., the attributes and methods of the class) are indented beneath the class declaration. In the example below, the class Customer is declared. By looking to the `__init__()` method, you will notice that the class consists of the following attributes: `firstName`, `lastName`, `phoneNumber`, and `address`. The constructor method is used to set the values of these attributes by using the `fNameP`, `INameP`, `phoneP`, and `addressP` parameters. The class consists of two other methods: `changePhone()` and `changeAddress()`.

```
class Customer:  
    #This is the class constructor method.  
    def __init__(self, fNameP, INameP, phoneP, addressP):  
        #These are the attributes of the Customer class contained inside the __init__ method  
        #The parameters above are being used to set the values of the class attributes  
        self.firstName = fNameP  
        self.lastName = INameP  
        self.phoneNumber = phoneP  
        self.address = addressP  
  
    #Below are two methods of the class: changePhone() and changeAddress()  
    def changePhone(self, phoneP):  
        #logic to change the phone number goes here  
  
    def changeAddress(self, addressP):  
        #logic to change the address goes here
```

## ***Using a class to create objects***

The code in the previous section simply creates the class template from which objects are instantiated. The code in the previous section would not do anything if you were to run it. To make the code do something, you must instantiate an object from the class. To instantiate an object from a class, you simply need to call the name of the class to access the constructor method. Objects are often stored in variables so that they can be accessed and manipulated through the class methods. Assume you have created the following class:

```
class Customer:
```

```
#This is the class constructor method.  
def __init__(self, fNameP, lNameP, phoneP, addressP):  
  
    #These are the attributes of the Customer class contained inside the __init__ method  
    #The parameters above are being used to set the values of the class attributes  
    self.firstName = fNameP  
    self.lastName = lNameP  
    self.phoneNumber = phoneP  
    self.address = addressP
```

```
#Below are two methods of the class: changePhone() and changeAddress()  
def changePhone(self, phoneP):
```

```
    #logic to change the phone number goes here
```

```
def changeAddress(self, addressP):
```

```
    #logic to change the address goes here
```

To instantiate the class, you will need to call the name of the class. In the case of this class, the constructor method (i.e., `__init__()`) has four parameters. You must pass in arguments for these parameters. You can create multiple distinct Customer objects from the same Customer class. The example below shows how this is done:

```
customer1 = Customer("Jamie", "Doe", "555-555-5551", "123 Fake St.")  
customer2 = Customer("Taylor", "Doe", "555-555-5552", "435 Fake St.")  
customer3 = Customer("Ryan", "Doe", "555-555-5553", "789 Fake St.")
```

## Calling class attributes and methods on an object

Once you create a class and instantiate an object from the class, you can begin to manipulate the object using the attributes and methods of the class. To do this, you simply need to remember the following syntax:

- `object.attribute` will return the value stored in the attribute requested
- `object.attribute = "value"` will set the value of a public attribute. Private attributes must be set with setter methods.
- `object.method()` will execute the named method belonging to the class

Building on the previous example, the example below shows how to use attributes and methods on an object:

```
customer1 = Customer("Jamie", "Doe", "555-555-5551", "123 Fake St.")  
customer1.lastName = "Doe-Rei" #change the value of the lastName attribute from Doe to Doe-Rei  
customer1.changePhone("555-555-5557") #call the setter method changePhone() with new argument  
print(f"My name is {customer1.firstName} {customer1.lastName}")
```