USING PYTHON FOR INTRODUCTORY BUSINESS PROGRAMMING CLASSES

Michael E. Ellis, University of Central Arkansas

Geoffrey Hill, University of Central Arkansas

Carla J. Barber, University of Central Arkansas

ABSTRACT

While no longer considered a core course, programming is an important part of many business school information systems (IS) programs. The choice of a language for introductory courses is an important one, and Python has become a good choice in recent years. Python is now considered the most popular programming language, largely due to its clean syntax and built-in functionality, plus it is in high demand by employers of our graduates. In this paper, we discuss the reasons for Python's rise in popularity and how those factors make it suitable for introductory IS programming classes in the business school. We also describe its use in our own introductory class within the context of the IS 2010 Curriculum Guidelines.

Keywords: Python, pedagogy, programming, business course, scripting language

INTRODUCTION

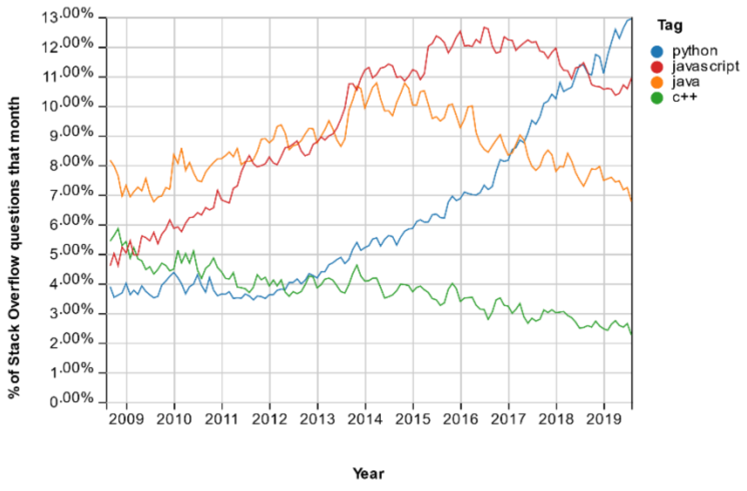
The 2010 IS Model Curriculum identifies programming as an elective for Information Systems (IS) programs. Removing programming as a core topic was intended to increase the flexibility of the curriculum guide to non-business IS programs (Topi et al., 2010). However, this does not preclude the inclusion of programming and application development as a featured component of IS programs. This appears to be a common practice since most IS programs consider programming to be an important part of their course offerings (Bell, Mills, & Fadel, 2013). Indeed, it may be argued that due to the rise of analytics and data science, the inclusion of programming is as important a skill as it ever was for IS programs.

Once the choice is made to include programming in an IS program, the question of what programming language (or languages) to teach becomes an important consideration, especially for the first programming course (De Raadt, Watson, & Toleman, 2002; Gries, 1974; Kölling & Rosenberg, 1996). For a time, specific teaching-centric languages such as Basic and Pascal were used for introductory programming courses. The shift to object orientation led to the rise of Java as a popular language for introductory programming courses. However, Python has been growing in popularity as the language of choice for introductory programming courses in computer science programs, taking the top spot from Java in 2014 (Guo, 2014; Robinson, 2017).

Not only has Python seen increased usage in academic programs, but it also has experienced great success in industry. Python is ranked third on the TIOBE index as of September 2019 (TIOBE - The Software Quality Company, 2019). This index is comprised of a proprietary rating based on the number of engineers, courses, and third-party vendors as derived from popular search engine rankings and results. Python also recently passed JavaScript as the highest-ranked language among the major programming languages (JavaScript, Java, Python, C++) in terms of keyword popularity on the Stack Overflow website ("Stack Overflow Trends," 2019). The rapid growth in the popularity of Python on the Stack Overflow site is shown in Figure 1 below.

The combination of applicability in academic computer science and industry settings makes Python an interesting new option for adoption in business programming courses. In this paper, we discuss how Python fits the needs of business programming classes. We begin by further discussing the rise in popularity of Python in both industry and academics. We continue with a discussion emphasizing the importance of programming in IS education and the differences between Computer Science and IS programming courses. We finish the paper describing the pros and cons of the Python language within the context of the topical guidance and learning objectives provided by the IS 2010 Curriculum Guidelines ("the Guidelines," Topi et al., 2010). Specifically, we discuss and highlight the effectiveness of Python as it applies to the seven learning objectives for the application development course identified in the Guidelines within the context of our program's introductory programming course, while also providing examples of active learning exercises highlighting the appropriate topics.

Figure 1. Stack Overflow programming language popularity ("Stack Overflow Trends," 2019)



THE RISE OF PYTHON

The choice of programming language for introductory programming courses in Computer Science (CS) has changed over the years. Fortran gave way to PL/I, which in turn was replaced by Pascal. Pascal was ousted by C++ in the mid-1990s and was later replaced by Java. (Siegfried, Siegfried, & Alexandro, 2016). Now, in many programs, Python has replaced Java.

The rise of Python as the language of choice for introductory programming courses in CS (commonly referred to as CS1 courses) is well documented. In July 2014, it was declared the most common language for CS1 courses because of its adoption in CS programs at top universities (Guo, 2014). The reasons for this increased use are many, but instructors often cite the language's "clean and simple syntax" (Goldwasser & Letscher, 2008, p. 1), which allows students to concentrate on concepts rather than the minutiae of the language. Python uses indentation and minimal punctuation to designate related blocks of code. Python also doesn't require students to declare variable data types upon initialization, end lines of code with a semi-colon (;), or use other similar syntax utilized by Java and C++. A serious concern when using Java in introductory classes, for example, is that "simple Java programs aren't simple" (Hunt, 2015, p. 173) because of the amount of overhead required for the most basic functionality.

Yet Python also has the high-level data types (Dierbach, 2014) needed to write powerful programs. Well-known applications like Instagram (McCracken, 2015) and YouTube (The Python Software Foundation, n.d.) rely on Python code to make them work. It is also robust enough to be proposed as an appropriate tool for learning object-oriented programming concepts (Goldwasser & Letscher, 2008; Miller, Settle, & Lalor, 2015). Python appears to meet most needs of an introductory programming course.

Python is not perfect, however. One concern for CS1 courses is that there is no true array support in Python. Lists and dictionaries – two of Python's often used data structures – implement most of the functionality of arrays but do not include the ability to allocate memory like a true array (Hunt, 2015). There is also concern that Python may be of little utility for CS students beyond its use in CS1 (Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM), & IEEE Computer Society, 2013). Many CS programs use Java or C++ in their second programming course. But as with learning anything, each programming language has its own set of rules, and switching between them has its own learning curve. At least one program (Hunt, 2015) has returned from Python to Java in CS1 because of the amount of time necessary in the second CS course to transition from Python to Java in order to cover more advanced CS-related concepts.

TEACHING PROGRAMMING TO BUSINESS STUDENTS

While there appear to be some programs that have business school IS students take the same introductory programming classes as computer science students (Siegfried et al., 2016), in our experience that is not the case. Still, all the reasons for using Python in CS1 classes also apply to introductory business programming classes, but there are other issues to consider as well based upon their expected career trajectories. When looking toward the future of the IS field, the use of Python for introductory programming classes empowers future curriculum decisions. More advanced use of Python is usually about analyzing data or automating tasks. That is a different path for Python coders than those with expertise in languages like Java. With the growth in data analytics, there is an increasing number of tasks for which Python is used in business that can be taught in upper-division business classes.

Are you teaching students expected to graduate with in-depth knowledge of a programming language so they can be placed into full-time coding positions? That may be a normal expectation for CS majors, but with most IS students, this is simply not the goal. Programming instructors in school of business IS classes typically look for ways to impart technical knowledge to students without getting too bogged down in the details. To be sure, the details are important, and we have graduates hired as programmers; however, in our experience, the goal of business programming classes is typically to introduce broader concepts to students to allow them to be able to communicate with programmers. The in-depth technical knowledge required for the level of expertise needed as a full-time programmer is beyond the scope of the typical business programming class. As the Guidelines state, students in these classes are expected to "learn the basic concepts of program design, data structures, programming, problem-solving, programming logic, and fundamental design techniques for event-driven programs." (Topi et al., 2008, p. 403) They simply do not need to learn to manipulate memory blocks, for example.

That lack of machine-level control available to Python programmers can be a problem for CS students, however. The CS 2013 guidelines acknowledge that there might be issues with using one of the "more managed languages" (Joint Task Force on Computing Curricula et al., 2013, p. 43) like Python. They may help the learning process while simultaneously disconnecting the student from how their code operates at the machine level. Another concern is that the languages that make it easier for students to learn programming basics may be useless to them in subsequent CS classes. Neither is a problem in IS in our experience.

As a scripted language, Python frees students from the need to perform a variety of manual tasks (such as compiling) often regarded as tedious and unclear as to their function (Alzahrani, Vahid, Edgcomb, Nguyen, & Lysecky, 2018). Python's syntax is clear and simple (Perkel, 2015), making it a popular choice for an educational setting. Additionally, Python includes support for a variety of graphics toolkits including the time-honored turtle graphics (Gaddis, 2017) that allows for combining visualizations with programming concepts, allowing students another avenue towards understanding.

With the focus of IS programs on those "fundamental concepts and models of application development" (Topi et al., 2010, p. 403), Python is well suited to the business IS curriculum. As a language guided by a single governing organization to maintain consistency and currency, supported and extended by a vibrant community of third-party developers, and yet syntactically easy to learn, Python provides the opportunity for business schools to provide academically and business-relevant programming experiences to their students.

Students who want (or need) to take a programming class but may not intend to write code in the future also benefit from using a language like Python. Learning programming teaches problem-solving and logical thinking skills. We also see students in a required programming class who find they enjoy writing code once they get some experience with it. They find they want to investigate topics like mobile app creation and data mining, which can require specialized programming environments of their own. Learning programming fundamentals in a simple, straightforward language like Python can provide a flexible introduction to the concepts for these students. These students receive little value from learning the intricacies of a more complex language like Java or C++. These more complex languages will often convince that type of student they have no business writing code and put an end to their exploration of programming before it really begins.

That is not to say Python is only good as a beginner's language. Python is, after all, listed as one of the skills with high demand from employers in the United States and Canada in the 2019 Robert Half Technology Salary Guide (2018). Many of our students have heard these types of statements and wish to learn more. Positions highlighted by the Robert Half Technology 2019 Salary Guide include business intelligence analysts, database developers, and systems administrators. Students pursuing these positions can all benefit from the acquisition of programming skills.

The explosion in the amount of data available to business analysts means that more people outside software development need fundamental programming skills for tasks like acquiring, cleaning, and analyzing data. These tasks can be performed in any language, but a simple language like Python allows analysts to focus on the data rather than the code. An ecosystem has grown around Python to make these analytic tasks easier to perform. The Python Package Index (PyPI: at https://pypi.python.org/pypi) contains a rapidly growing collection of thousands of free to use projects containing over two million packages (as of 9/26/2019) of specialized Python code, many of which are intended for data-centric tasks. Some well-known packages and their applications are

- BeautifulSoup for Web scraping,
- SciPy for math, science, and engineering work,
- Pandas for data analysis and modeling, and
- Django for Web programming.

Table 1. Growth of Python project availability (“PyPI · The Python Package Index,” n.d.)

Date	Number of Projects Available
1/13/2017	96,550
7/26/2018	147,158
3/4/2019	170,729
9/26/2019	197,715

Some coding ability is needed to use these packages but not at the same level as the coding ability needed to create them. Learning Python at the introductory level gives business students a head start when it comes to using these more advanced applications as undergraduates, graduate students, or as they begin their careers. IS programs are increasing the number of data science and analytics programs (Aasheim, Williams, Rutner, & Gardiner, 2015; Mills, Chudoba, & Olsen, 2016; Wymbs, 2016) to meet the demand for these skills in the workplace.

APPLICATION DEVELOPMENT COURSE LEARNING OBJECTIVES

The Guidelines include a general description of the Application Development elective course, and learning objectives the authors felt were important for the course (Topi et al., 2010). These objectives can be met using Python as with any other mainstream programming language. We will use the design of an introductory Python programming course to address how these objectives can be met.

Table 2. Application Development Course Learning Objectives (Topi et al., 2010)

LO-1	Use primitive data types and data structures offered by the development environment.
LO-2	Choose an appropriate data structure for modeling a simple problem.
LO-3	Understand basic programming concepts.
LO-4	Write simple applications that relate to a specific domain.

LO-5	Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures, and the definition of functions.
LO-6	Test applications with sample data.
LO-7	Apply core program control structures.

General Structure of the Introductory Course

All programming language classes (Java, Python, Visual Basic (VB), and COBOL) previously taught in our department began with the assumption that students had not programmed before. While this is indeed true of many students, it was not true of a sizable portion. In many cases, a student was taking a second programming course in the department and seeing the same material in a different language. While repetition can be beneficial for students, we felt we had an opportunity to give our students a better programming course experience. This led to the creation of a common introductory programming class that is a prerequisite for the others.

This introductory class utilizes Python to teach business students the basics of variables, looping, decisions, and other fundamental programming concepts. This approach allows the language-specific classes that follow to do a quick review of the fundamentals, then spend more time exploring the features and characteristics that make that language so valuable. For example, after a few classes of review, the original Python class can now go further into topics like data cleansing, Web scraping, and even some basic analytics.

Classes are conducted in a computer lab classroom. Rather than lecturing on subject material, the instructor creates programs during class meetings that are projected live at the front of the room. Students simultaneously build the programs on the lab computers, or their personal laptops if they prefer. Any necessary conceptual information on the day's material can then be discussed within the context of the program being written.

Students are also given time for in-class exercises. Very short tasks covering concepts and code just completed as a class are given to students to work on briefly. These exercises give students a chance to get a first experience with writing the code before they must use it to complete a homework assignment. There are five to seven programming homework assignments in the course, depending upon the instructor, and three exams covering concepts and code interpretation.

Since the main purpose of this course is to get students on a solid footing of programming fundamentals, it assumes no programming experience. That assumption is usually but not always true, but the goal is to get everyone to the point where they are competent enough with simple programming tasks that they can be successful in other programming courses if they should decide to pursue that path or at least have the fundamental skills to continue to learn on their own after graduation. Most of these skills are embodied in the learning objectives found in the Guidelines. We describe how this course starts with basic programming concepts (LO-3 from Table 2) and increases the level of complexity as the students progress through the semester, meeting the learning objectives as they go.

The Basics: Learning Objective 3

The phrase “basic programming concepts” is used regularly, but there appears to be no specific definition of what it entails. It can refer to very general concepts or concepts that would manifest in specific programming skills. General concepts like defining what a computer program is and the logic of problem-solving with a computer are important to discuss but do not require code writing to explore. They can be demonstrated before students even turn their computers on. For example, one of the authors uses the preparation of boxed macaroni and cheese as an example of how to write instructions for a computer. It is an activity most students have performed, so they are comfortable discussing it. They are surprised, however, to discover how what they thought was a simple process is much more complicated when all the assumptions inherent with an anticipated human preparer have to be translated into the detailed instructions required by a computer.

Other basic concepts, like using variables to store values for further manipulation, are conceptual in nature but require a specific language to demonstrate how the concept works. This is an example of a concept we would expect to see manifest in specific programming skills acquired by the student. We use “basic programming concepts” to refer to those concepts related to the specific programming skills listed in Table 3. In our program, we have previously identified them as being important for students to acquire no matter the underlying language. At a conceptual level, they are all language agnostic but must be taught in a specific language to demonstrate how they are implemented.

Table 3. Basic programming concepts (as defined by the authors)

Creating and using simple variables	Simple keyboard input and screen output
Creating and using named constants	Writing comments
Basic mathematical operators	Naming standards and conventions
Performing simple calculations	Using functions
Reserved words	

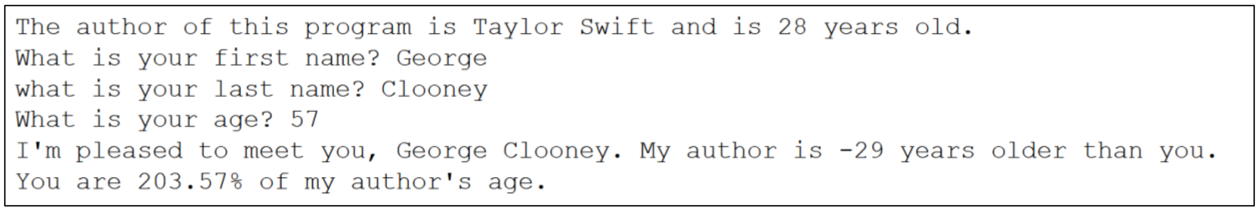
The nature of Python makes it straightforward to cover all these concepts. We begin talking about reserved words, writing comments, and following all naming conventions when we first begin to write code. The in-class coding done in tandem with the instructor includes these concepts, and subsequent assignments require them for full credit. Examples used in class repeat these concepts as we progress through the semester.

The first homework assignment incorporates the remaining basic concepts. In that assignment, students are tasked to do the following:

- Assign their own first name, last name, and age to named constants.
- Collect from the user values for first name, last name, and store them in appropriately named variables.
- Calculate the difference between the two ages.
- Display greetings to the screen featuring the age difference and the user’s age in terms of the percent of the programmer’s age.

An example of the required output for this assignment using celebrity data is shown below in Figure 2.

Figure 2. Example output for basic programming concepts assignment (Note: the values "George," "Clooney," and "57" are input by the user.)



```
The author of this program is Taylor Swift and is 28 years old.
What is your first name? George
what is your last name? Clooney
What is your age? 57
I'm pleased to meet you, George Clooney. My author is -29 years older than you.
You are 203.57% of my author's age.
```

Getting keyboard input from users and displaying to the screen requires the use of functions. The Python functions used to accomplish these two tasks (*input* and *print*, respectively) are also examples of reserved words. The nature of reserved words, functions, and how functions should be used is covered during previous classes as part of the in-class coding and reinforced in this assignment.

The student must also write the code to perform very basic calculations using values stored in named constants and variables. When data like that of the Figure 2 example is used where the user's age is greater than the programmer's age, the program simply displays a negative result for the number of "years older than you" age difference. The somewhat odd nature of that portion of the output helps introduce why we need to use decision structures.

Data Modeling: Learning Objectives 1 & 2

Students use the primitive variable types built into Python, beginning with the code written during the first class meeting. Examples used to introduce students to the basics of programming must include declaring and assigning values to variables to enable further activities. Where Python differs from other languages in this area is that Python does not require the explicit declaration of variables. Variables are defined and typed based upon the value assigned to them when they are first used. Students with experience in other programming languages are often uncomfortable with not having first to declare and type a variable, but we have found that removing that step for new programmers allows them to concentrate on what they are trying to do with the variables rather than the variables themselves.

In some cases, students must convert their data from one type to another. A common example involves the difference between numeric strings and numbers. Numeric strings are made from number characters, but they are not used in calculations. A social security number or a telephone number are examples. Another example is in the first homework assignment. Python's *input* function is used in the first homework assignment to accept keyboard input from the user. Every value returned by *input* is a string value, regardless of the characters it contains. To do the simple calculations required by the assignment, students must convert the user input numeric string values into numbers. This gives them further experience with the primitive data types and the relationship between them.

Converting numeric strings into numbers also introduces why it is important to specifically choose the appropriate data structure for the task at hand. Calculations require numbers and don't work with strings. Common numbers used in business programming, like product prices or hourly wage, need to be float variables because they have important decimal components. Product quantity is usually an integer value because you can't have a partial unit of product – unless, as with bulk food items and numerous other examples, you can.

The implied variable declaration and typing used by Python allows the instructor to focus on the larger issues involved in these examples. Students can learn how to determine the variable type they need for a particular problem without the added complexity of explicit declaration statements.

Control Structures: Learning Objective 7

The seventh learning objective (Topi et al., 2010) refers to teaching the application of core programmatic control structures. Python supports the expected collection of logic control structures associated with structured programmatic design, including sequencing, selection, and repetition. This support fulfills the base requirement of this stated learning objective for the inclusion of these structures within the language. The prior discussion highlighting typical IS programs' focus on applied concepts while downplaying the more technical programming aspects are well supported by Python's language implementation surrounding these fundamental control structures is of great importance to this objective. We suggest that Python's relatively simple implementation of these controls better positions itself to be the language of choice for teaching programmatic concepts in a business school.

To exemplify this, we turn to a simple example shown below in Figure 3 and Figure 4. This is the same simple program written in both Java and Python. As evidenced by the Java example, a discussion of the selection control structure must be preceded or accompanied by discussions of concepts such as classes, libraries, methods, arguments, data types, object instantiation, closing open resources, along with other non-obvious concepts. By way of contrast, the more easily understood Python equivalent code allows the beginning programming student to interpret and understand the programming code while keeping the learning focused on the selection control structure. The equivalent Python code needs to be preceded or accompanied by a much smaller list of predicate knowledge such as indentation, syntactic language requirements, and a few discrete language-specific function operations such as print(), input(), and int().

Figure 3. Java Selection Example

```
import java.util.Scanner;

public class SelectionExample {

        public static void main(String[] args) {
                Scanner input = new Scanner(System.in);
                System.out.println("Please enter your age: ");
                int age = input.nextInt();
                if (age >= 18) {
                        System.out.println("You are old enough to vote in the U.S.");
                } else {
                        System.out.println("You are not old enough to vote in the U.S.");
                }
                input.close();
        }
}
```

Figure 2. Python Selection Example

```
age = int(input("Please enter your age: "))
if (age >= 18):
    print("You are old enough to vote in the U.S.")
else:
    print("You are not old enough to vote in the U.S.")
```

Complete Applications: Learning Objective 5

The ability of students to successfully implement discrete programmatic concepts into holistic, multiple-concept constructs is obviously of tremendous importance to any programming related discipline. This is the focus of the fifth learning objective (Topi et al., 2010). Here too, Python is very well situated to support this learning objective through its comparatively simple language requirements of the many discrete concepts appropriate to an introductory programming course. As exemplified below in Figure 5 and Figure 6 below, the Python code is much simpler when compared to its Java counterpart. This facilitates focusing upon the concepts and how the beginning programmer can manipulate the language requirements to accomplish a comprehensive goal of a programming assignment requiring the concurrent implementation of discrete concepts such as input/output, control structures, and functional definition.

Figure 5. Java Multi-concept Example

```
import java.util.Scanner;

public class SelectionExample {

        private static boolean isEligible(int age) {
                if (age >= 18) {
                        return true;
                } else {
                        return false;
                }
        }

        public static void main(String[] args) {
                Scanner input = new Scanner(System.in);
                System.out.println("Please enter your age: ");
                int age = input.nextInt();
                if (isEligible(age)) {
                        System.out.println("You are old enough to vote in the U.S.");
                } else {
                        System.out.println("You are not old enough to vote in the U.S.");
                }
                input.close();
        }
}
```

Figure 6. Python Multi-concept Example

```
def is_eligible(age):
    if age >= 18:
        return True
    else:
        return False

age = int(input("Please enter your age: "))
if (is_eligible(age)):
    print("You are old enough to vote in the U.S.")
else:
    print("You are not old enough to vote in the U.S.")
```

Business Context: Learning Objectives 4 & 6

Many sources of introductory programming material for students use generic examples, and we do too. For example, this introductory course uses assignments from the Gaddis (2017) text that rely on the well-known turtle graphics module found in the Python standard library ("The Python Standard Library," 2017). These problems, which require students to programmatically move the cursor around the screen to create drawings, both engage students and give them practice with conditional control, repetition, and other concepts represented in the learning objectives from the Guidelines. But beginning business programming students often do not make the connection between example programs that check a number to see if it is a prime number and business-related problems if they do not see that connection in class. We use problems featuring business-oriented calculations whenever possible.

All programs used in this course, whether short in-class examples or full homework assignments, come with test data. The reason to provide test data is twofold. First, we are trying to make students understand the importance of testing their code to ensure its accuracy. The only way to do that is with data that produces known results or by creating data that generates easy to verify results. For example, if we want to check a sales tax calculation of 8.5%, we can apply it to a $100 purchase. That way, we know the result should be $8.50. If the result is something else, we know we have a problem in the code and can investigate.

The second reason to provide test data is so students know when they have solved the problem. Imagine asking someone to build a table when they have never seen one. How high does it need to be? How long? How many legs? What shape? Students new to programming need to know what the target looks like so they have a chance to hit it. If they are not given test data, they tend to get frustrated and quit, or they continue to solve the problem well after it is solved. Neither is a positive outcome. With later assignments (and subsequent programming courses) it is perfectly reasonable to require students to create their own test data.

TOOLS USED

The choice of tools for the class, like textbooks or programming tools, is largely dependent upon the preferences of the instructor. We have adopted a common text for this class, but any introductory Python programming book will work if the instructor is comfortable with its pedagogical approach. The primary concerns we addressed during textbook selection were the order in which programming concepts were covered and the depth to which they were covered. Some books were thought to go into too much detail about how code works at the CPU level, for example. These books might be perfect for a computer science course but were felt to be inappropriate for the business programming course.

As with many other programming languages, Python code can simply be typed into a text file and executed. An Integrated Development Environment (IDE), however, provides mechanisms for executing code, language syntax validation, debugging functions, plus a host of other tools that can make it easier for beginning programmers to learn new concepts. Fortunately, Python is popular enough to have many contemporary options. We have used three different IDEs for this course, depending upon the instructor, and all have been successful. Each has its strengths and weaknesses, which might best be considered within the larger context of the pedagogical purpose of this course.

The simplest IDE option is the IDLE included within the base distribution of the Python development kit. Its simplicity of being pre-installed significantly enhances its usefulness in introductory programming courses. Our experience has shown students frequently encounter difficulty installing alternative IDE software. This occurs because IDE requirements often include multiple software package downloads, complex post-installation configuration, and installation of add-ons required for use with various course modules. However, the IDLE still provides a basic set of tools that is usually enough for the level of programming done in this course. For solid functionality with minimal complexity, the IDLE is a good choice of IDE.

However, the IDLE's simplicity can be overshadowed by its dearth of features when compared to other modern IDE software. One popular alternative is Eclipse. Eclipse, although requiring a separate installation of software in addition to the required Python development kit, is a full-featured IDE in use in industry and long recognized as one of the dominant IDE packages for production-level development across many different programming languages (Geer, 2005). Eclipse does perhaps suffer a bit from a comparatively steep learning curve. But it may be argued that overcoming this learning curve in the academic environment enhances the programming student's value to potential employers in that they are also generating understanding and familiarity with professional-level tools while simultaneously learning the focal programmatic concepts. For the student who wants to be a full-time programmer, Eclipse is the better choice.

As previously discussed, advanced uses of Python typically involve data analysis tasks. A popular choice of IDE in this area is the Jupyter Notebook. Jupyter Notebook is an open-source browser-based application you can use to create and share documents that contain live code, equations, visualizations, and text (Driscoll, 2019). The Jupyter Notebook name comes from the core supported programming languages it supports, **Ju**lia, **Pyt**hon, and **R,** but the Notebook now supports over 40 languages (Project Jupyter, n.d.). It uses individual containers called "cells" to hold its contents. Markdown cells hold notes or other documentation, while code cells hold the Python code. These individual cells allow the code to be executed in small pieces while preserving the results of previously run code. This is a very useful feature for beginners because any new errors are limited to the last cell executed, making the debugging process much easier.

It is also a great tool to use in a lab classroom as it allows the instructor to use the Notebook's markdown cells to include notes on concepts to be covered in class. The instructor can then share the notebook with students and use it as the basis for any programming examples completed during class. Students are able to insert a cell as needed and enter the code for a particular example while following the instructor's directions. At the end of the class session, students have all the code and notes on a topic in one file in a format that resembles a completed analysis project. For the student in a data analytics or data science program, the Jupyter Notebook is a good choice of IDE.

LESSONS LEARNED

Now that we have been operating under this programming curriculum model for several semesters, we have learned a few lessons that we have used to adjust some aspects of the course. In general, these lessons fall under the categories of how students try to get help, issues with students who have previous programming experience, and group projects.

Students Getting Help

There appears to be a tendency among current students to spend an hour looking for a solution online when they could find what they need in their textbook in five minutes. Because Python is currently deployed in two major versions (2.x and 3.x), online search results can also be problematic when students locate a solution using the wrong version of Python. The course is intended to use the current 3.x version of Python. The 2.x version is still available because of legacy programs in use, but it will not be supported beyond January 1, 2020 ("PEP 373—Python 2.7 Release Schedule," n.d.). Many code snippets and help answers found online and in some books about Python use the 2.x version of Python. Students can feel like they have made an important discovery when solving a programming issue when, in fact, they have discovered commands that will soon become obsolete or may not work at all in version 3.x. They are also surprised when their working homework assignment program receives a very low score.

Another issue is that it is easy for students to discover techniques and syntax online that are well beyond the intended scope of the class. For the occasional student, that is not a problem, but for most, it only serves to increase their confusion. They may spend an inordinate amount of time trying to figure out what to do with something they find online that may seem to do what they want but instead raises more questions for them. This problem is also caused by the tendency to go online first when looking for help.

Another aspect of this tendency – and certainly not one specific to Python – is that students become distracted by trying to make their programs produce "the answer" instead of using the textbook and other class resources to work out the logic of a solution. In many cases, they can miss the entire point of the assignment. When unable to work out the logic of looping through several pieces of data, for example, students in this class have hardcoded a solution that generates output that looks like the example output they were provided but does nothing else. They got "the answer" but missed the point of the assignment entirely.

To address these issues, subsequent offerings of the class have limited students to the scope of the textbook and class materials. Online resources can still be helpful to explain and illustrate concepts, but online content creators do not have the same motivations and goals that are embedded in the course design. Help material found online usually appears to be written *by* experienced programmers and *for* experienced programmers. Students new to programming can get lost in the complexity of an answer found online and become more frustrated. Limiting them to what is in the curated textbook helps reduce that unnecessary frustration.

Students with Programming Experience

Students with previous programming experience present challenges in a class intended for absolute beginners in any language. Many of the features of Python considered by most to be advantages can be seen by the experienced student as either simplistic or incomplete. We have found this either leads to inaccurate assumptions or frustration for these students.

The student who sees Python as simplistic begins to assume that it will take almost no effort for them to earn an “A” for the class. They are correct in thinking they have an advantage over other students, but the error is that they do not yet know what they don’t know about programming in Python. When they complete the first homework assignment in minutes, their flawed assumptions are confirmed, and they can often “check out” of the class, regularly skipping class and eventually doing poorly. This type of student can find themselves earning a “C” when they should be an “A” student.

For experienced students who get frustrated, it is helpful to point out the differences between Python and other languages whenever possible. One example is the previously mentioned Python method for variable declaration. Languages like C++ and Java require an explicit variable declaration, which involves commands associated with the intended variable name to create and type the variable. In Python, programmers provide the name and assign a value. Python handles the rest. Explaining what is going on behind the scenes with Python can often – but not always – alleviate some of the concerns of this type of student.

Group Projects

Programming can be (and is often taught as) a solitary activity, yet CS programs have acknowledged the need for programming students to get beyond the stereotype of the programmer as a loner (Heiner, 2014). The skills discussed by Heiner that students can develop in group work are plentiful: conflict resolution, general communication skills, learning to discuss technical ideas in plain English, and more. One of the common threads throughout most business school curriculums is the requirement of group projects. We regularly talk to employers who ask for their student hires to have experience working in groups.

VB is a very effective language to use for group projects. The emphasis on the “visual” with VB means there are usually several user screens that need to be created plus the code that runs behind them to manipulate data and generate the desired results. The tasks involved in creating these items can easily be split among group members and reassembled into a final product. We might also want business programming students to develop a VB program designed to solve a business problem. In our experience, most groups create a system intended to allow the user to make a sale of some kind. Again, the need for input screens, output, and databases, in addition to the code functionality make this type of project work in that context.

Most uses of Python, however, do not typically result in an emphasis on a GUI-type program. They tend to be stand-alone programs that complete a particular task. More complex projects could be undertaken, and by using features like custom functions could become feasible in a Python course. However, those concepts are beyond the scope of our course but could be used in a subsequent course with more experienced Python programming students. So, for now, the group project does not appear to work for this course.

CONCLUSION

Python has become the leading language in practice and the language of choice for introductory CS programming classes. The traits that make it attractive in CS also make it attractive as a first language in business IS programming classes. Other characteristics that make IS programming classes different (Siegfried et al., 2016) also lend themselves to the use of Python.

It is also a relatively easy language for programming course instructors to learn. Familiarity with any major programming language will make it simple to understand how Python handles the tasks taught in any introductory programming class. A little experience (and good notes!) make it straightforward to teach with Python for the first time.

REFERENCES

Aasheim, C. L., Williams, S., Rutner, P., & Gardiner, A. (2015). Data analytics vs. Data science: A study of similarities and differences in undergraduate programs based on course descriptions. *Journal of Information Systems Education*, *26*(2), 103–115.

Alzahrani, N., Vahid, F., Edgcomb, A., Nguyen, K., & Lysecky, R. (2018). Python Versus C++: An Analysis of Student Struggle on Small Coding Exercises in Introductory Programming Courses. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 86–91. https://doi.org/10.1145/3159450.3160586

Bell, C. C., Mills, R. J., & Fadel, K. J. (2013). An Analysis of Undergraduate Information Systems Curricula: Adoption of the IS 2010 Curriculum Guidelines. *Communications of the Association for Information Systems*, *32*(Article 2), 73–94.

De Raadt, M., Watson, R., & Toleman, M. (2002). Language trends in introductory programming courses. *Proceedings of the 2002 Informing Science+ Information Technology Education Joint Conference (InSITE 2002)*, 229–337. Informing Science Institute.

Dierbach, C. (2014). Python As a First Programming Language. *Journal of Computing Sciences in Colleges*, *29*(6), 153–154.

Driscoll, M. (2019, January 28). Jupyter Notebook: An Introduction. Retrieved November 1, 2019, from Real Python website: https://realpython.com/jupyter-notebook-introduction/

Gaddis, T. (2017). *Starting out with Python* (4th ed.). Pearson.

Geer, D. (2005). Eclipse becomes the dominant Java IDE. *Computer*, *38*(7), 16–18. https://doi.org/10.1109/MC.2005.228

Goldwasser, M. H., & Letscher, D. (2008). Using Python To Teach Object-Oriented Programming in CS1. *Innovation and Technology in Computer Science Education*, (June). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.156.8784&rep=rep1&type=pdf

Gries, D. (1974). What Should We Teach in an Introductory Programming Course? *ACM SIGCSE Bulletin*, *6*(1), 81–89. https://doi.org/10.1145/953057.810447

Guo, P. (2014, July 7). Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities. Retrieved October 12, 2015, from http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext

Heiner, C. (2014). Stages of Group Work in CS1. *Journal of Computing Sciences in Colleges*, *30*(2), 79–84.

Hunt, J. M. (2015). Python in CS1—Not. *Journal of Computing Sciences in Colleges*, *31*(2), 172–179.

Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM), & IEEE Computer Society. (2013). *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Retrieved from https://dl.acm.org/citation.cfm?id=2534860

Kölling, M., & Rosenberg, J. (1996). Blue—A Language for Teaching Object-oriented Programming. *ACM SIGCSE Bulletin*, *28*(1), 190–194. https://doi.org/10.1145/236462.236537

McCracken, H. (2015, June 23). Do The Simple Thing First: The Engineering Behind Instagram. Retrieved February 26, 2019, from Fast Company website: https://www.fastcompany.com/3047642/do-the-simple-thing-first-the-engineering-behind-instagram

Miller, C., Settle, A., & Lalor, J. (2015). *Learning Object-Oriented Programming in Python: Towards an Inventory of Difficulties and Testing Pitfalls* (No. 25). Retrieved from DePaul University website: http://via.library.depaul.edu/tr/25/

Mills, R. J., Chudoba, K. M., & Olsen, D. H. (2016). IS Programs Responding to Industry Demands for Data Scientists: A Comparison between 2011-2016. *Journal of Information Systems Education*, *27*(2), 131–140.

PEP 373—Python 2.7 Release Schedule. (n.d.). Retrieved September 27, 2019, from Python.org website: https://www.python.org/dev/peps/pep-0373/

Perkel, J. M. (2015). Programming: Pick up Python. *Nature News*, *518*(7537), 125.

Project Jupyter. (n.d.). Project Jupyter. Retrieved November 7, 2019, from Jupyter.org website: https://www.jupyter.org

PyPI · The Python Package Index. (n.d.). Retrieved September 26, 2019, from PyPI website: https://pypi.org/

Robert Half International. (2018). *2019 Robert Half Technology Salary Guide*. Menlo Park, CA.

Robinson, D. (2017, September 6). The Incredible Growth of Python | Stack Overflow. Retrieved August 1, 2018, from Stack Overflow Blog website: https://stackoverflow.blog/2017/09/06/incredible-growth-python/

Siegfried, R. M., Siegfried, J. P., & Alexandro, G. (2016). A Longitudinal Analysis of the Reid List of First Programming Languages. *Information Systems Education Journal*, *14*(6), 47–54.

Stack Overflow Trends. (2019). Retrieved September 28, 2019, from https://insights.stackoverflow.com/trends?tags=python%2Cjava%2Cc%2B%2B%2Cjavascript

The Python Software Foundation. (n.d.). Quotes about Python. Retrieved February 26, 2019, from Python.org website: https://www.python.org/about/quotes/

The Python Standard Library. (2017, September 19). Retrieved September 27, 2019, from The Python Standard Library—Python 3.3.7 documentation website: https://docs.python.org/3.3/library/index.html

TIOBE - The Software Quality Company. (2019, September). TIOBE Index. Retrieved September 28, 2019, from TIOBE Index for September 2019 website: https://www.tiobe.com/tiobe-index/

Topi, H., Valacich, J. S., Wright, R. T., Kaiser, K., Nunamaker, Jr., J. F., Sipior, J. C., & Vreede, G. J. de. (2008). Revising Undergraduate IS Model Curriculum: New Outcome Expectations. *Communications of the Association for Information Systems*, *23*(Article 32), 591–602.

Topi, H., Valacich, J. S., Wright, R. T., Kaiser, K., Nunamaker, Jr., J. F., Sipior, J. C., & Vreede, G. J. de. (2010). IS 2010: Curriculum Guidelines for Undergraduate Degree Programs in Information Systems. *Communications of the Association for Information Systems*, *26*(Article 18), 359–428.

Wymbs, C. (2016). Managing the innovation process: Infusing data analytics into the undergraduate business curriculum (lessons learned and next steps). *Journal of Information Systems Education*, *27*(1), 61–74.

QRBD

QUARTERLY REVIEW OF BUSINESS DISCIPLINES

November 2019

Volume 6
Number 3

A JOURNAL OF INTERNATIONAL ACADEMY OF BUSINESS DISCIPLINES
SPONSORED BY UNIVERSITY OF NORTH FLORIDA
ISSN 2334-0169 (print)
ISSN 2329-5163 (online)