Teaching Introduction to Programming Languages with a Database Twist*

Suzanne W. Dietrich
School of Mathematical and Natural Sciences
Arizona State University
Phoenix, AZ 85069
dietrich@asu.edu

Abstract

Programming languages are fundamental to the computing discipline. Students need to learn multiple programming paradigms as well as their underlying principles as part of their degree. A programming language course covering the imperative, functional, and logic language paradigms is typically a difficult course for students as they learn new ways of problem solving. This paper reports on the pedagogy and experience of teaching such a course in the context of a computing degree in a college of interdisciplinary arts and sciences with only CS2 (Java) as a prerequisite. The choice of Python, Erlang, and Prolog provides a sequence of languages that incrementally adds new features to be mastered by the students. The pedagogical approach uses a combination of skeletal notes, reading quizzes, practice exercises, and active learning to engage students in the learning process. For each language, a programming assignment supporting depth of learning uses a database approach to create relationships between the data and to answer realistic questions over the data.

1 Introduction

The computing B.S. degree program in our college of interdisciplinary arts and sciences provides an alternative degree to the conventional computer science degree offered in an engineering college. The lower-level courses include CS1 and CS2 in Java, Introduction to Programming Languages, Data Structures & Algorithms, Discrete Mathematical Structures, Brief Calculus, and Statistics. The upper-level courses emphasize databases, networks, and cybersecurity. There is an experiential learning degree requirement, consisting of internships and/or research. The most recent introduction to the lower-level core courses was the Introduction to Programming Languages course with CS2 as its only prerequisite. Adding this course provided a programming language component that was missing from our degree and enhanced pathways for transfer students from Computer Science at the local community colleges.

The design of the course was based on two main components. The first being the Computing Curricula 2013 [15] and the second was the coverage in the corresponding course in the engineering college of our university. The topics covered for the basic principles and concepts of programming languages were essentially derived from these sources. However, the choice of programming languages to use for the coverage of the imperative, functional, and declarative paradigms should be based on unit-specific objectives.

Our unit within the college offers bachelor degrees in computing, applied mathematics, statistics and the natural sciences. The applied mathematics and statistics students are required to take CS1 and CS2 as part of their degree programs. Some of these students double major or minor in computing. Due to the popularity of Python and its applications within the unit, Python is the imperative language choice. The selected functional language is Erlang, based on its applicability in industry for 24x7 Web applications and app development. Prolog, Programming in Logic, is the canonical choice for a declarative, logic language.

This paper reports on the experience of the resulting design of an Introduction to Programming Languages course that emphasizes connections with databases in programming assignments. Section 2 provides a brief overview of related work. The pedagogical approach used in the course across all three languages is presented in Section 3, along with a semester schedule of topics and assignments. Sections 4, 5, and 6 discuss the resources used for the languages along with the specifics of the programming assignments. Section 7 reflects on the experience.

2 Related Work

A background literature review searching for teaching the programming language paradigms course found many older publications on the topic. One of these [22] provided an excellent overview of known issues with language choice for the course, including: prior experience of the student, the choice of the CS1/CS2 language, the use of pure versus hybrid languages to illustrate each paradigm, and the use of a universal language, such as Ada, to illustrate all of the paradigms. Another paper provided additional reasons "to better understand the difficulty of designing an effective curriculum" [5] for such a course, such as object confusion, breadth versus depth, effect of scale, indoctrination (tools influence problem-solving approach) and desensitization (work around shortcomings of a tool). Most computer scientists would probably agree that the design of the programming language paradigms course is difficult.

Some additional, more recent papers on the subject emerged. One alternative approach explored a methodology where students who were computer science majors would not be mere users of the programming languages but focus on the concepts as if they would become a programming language designer [16]. Another approach emphasized that "doing is better than seeing", although it is more difficult, by designing a one-semester course where students combined the language paradigms in the development of a compiler for a non-trivial subset of a functional programming language [11]. Another study recognized the difficulty of students even becoming users of a programming language in such a course and introduced short, practice exercises that improved learning outcomes for the topics addressed by the practice and increased the students' confidence [1].

The approach presented in this paper also recognizes the many challenges in designing such a programming language paradigms course. The following section provides an overview of the various pedagogies employed to help students learn how to use the programming paradigms, which also includes the incorporation of shorter, practice exercises.

3 Overview of Pedagogical Approach

Table 1 provides a weekly semester schedule for the programming languages course. The basic principles notes cover grammars, parsing and derivation trees, the compilation process, and compilation versus interpretation. The concepts of programming languages notes detail names, bindings and scope; data types; expressions and assignment statements; statement-level control structures; and subprograms (parameter passing and activation records). The basic principles and concepts of programming languages notes are based on coverage

from the following texts: [4], [8], [17], [18]. Each language is introduced in the context of these fundamental concepts. The course employs various pedagogical approaches, including skeletal notes, short programming exercises, reading quizzes, and active learning exercises. An overarching goal of the pedagogy is to engage students in active participation in their learning as well as to reflect on what they have learned.

Table 1: Semester Schedule

Week	Topics	Assignments
1	Language Overview in Degree Program Basic Principles	BNF WebQuest
2	Concepts of Programming Languages	
3	Python: Strings, Decisions, Loops	Reading Quizzes Codingbat
4	Python: Tuples, Lists, Sets, Dictionaries	Reading Quizzes
5	Python: Modules, IO, Exceptions	Reading Quizzes Review Exercises
6	Python: Classes and Inheritance	Python Quiz Assign Program
7	Active Learning	BankAccount in class
8	Event-based with TKinter	TKinter in class
9	Midterm Review	MIDTERM
10	Erlang: Tuples, Lists, Strings	Codingbat
11	Erlang: List Processing, Maps, Files, CSV	Assign Program
12	Erlang: Review Exercises	Erlang Quiz
13	Prolog: Overview and Database Connection	Codingbat
14	Prolog: findall and Review Exercises	Assign Program
15	Higher-order programming: All 3 languages	Review for Final

Skeletal notes is a term that references a set of incomplete lecture notes, where students are responsible for filling in some of the parts. For each language and for each data structure, the class notes provide an overview of the data structure, its fundamental operations and methods. There are a couple of examples in the notes that the instructor illustrates on the language in its IDE. Each data structure includes a skeletal notes page that contains a table

of 6-8 problems over the data structure with space for students to write the expression/code to answer the question and to record the response from the execution of the code. Before the start of the next class, students must complete the notes page and submit their attempt on the course management system for participation points. They are allowed to work with another student in the class and they are asked to document that collaboration. The day that the notes are due, there is a brief class discussion answering any major questions students may have. The next class, after reviewing the students submissions, common mistakes are discussed and corrected.

Practice is an essential component of learning programming languages. Once the fundamentals of the language are covered, the students are introduced to the codingbat.com site. In class, the inOrderEqual and twoAsOne codingbat examples provide students with illustrative samples. Students are then given an assignment using additional exercises from codingbat.com to explore the language. The examples chosen (countHi, in1to10, alarmClock, repeatSeparator, mixString) provide students with practice on string processing, comparison operators, nested comparisons, and iteration/recursion. The students are given a document that has the description from the codingbat site, and includes space for the student to record the code that they ran on codingbat and what they learned from writing that code. They are also asked to include a screen shot of the test cases that passed. The students are instructed that it is not the answers that are important but the journey of learning. The students submit the document for participation credit.

There is a programming assignment on each language to provide a depth of learning. Each semester a different data set from [21] is selected to provide realistic but cleaned data to use in the assignment. Students write programs in each language to process the data and answer questions over the data. More details appear later in the paper with respect to the programming assignment in each language.

After covering the language and before a quiz, students have an in-class exercise to complete a set of "Review Exercises". These designed exercises have students apply different facets of the languages:

1. letter_count: Return a dictionary/map of the count for each letter appearing in the parameter string.

2. both_strings: Return a string that includes the characters appearing in both parameter strings.

3. repeats_num: Determine whether the sequence of numbers has a repeated number.

4. order_summary: Given a list of tuples representing an item, price, and

quantity on an order, return a tuple with the total cost of the order and the total number of items.

Students are broken down into groups of 4-5 and assigned one of these exercises to work out as a team. Then volunteer groups are asked to record their answer on the white board for class discussion. Pictures are taken and posted on the course management system. Students are strongly encouraged to code the review exercises themselves and verify.

For quizzes and exams, students are provided with a one-page, two-sided reference sheet for the syntax of each language. For Erlang and Prolog, this reference sheet also includes examples of code, such as inOrderEqual, twoAsOne, and list processing. When the assessed midterm is returned in class, students fill out a questionnaire, also known as an *exam wrapper* [12], to reflect on their preparation, what they would do differently for the next assessment, and provide feedback to the instructor. These are collected for instructor feedback and returned to students for the final exam.

4 Python

Our CS1 and CS2 courses use the Java programming language. The goal for learning Python is for students to have the same level of knowledge as Java. Due to the diverse backgrounds of students and the amount of transfer students, an interactive Python textbook has been adopted [10] so that students can fill in their knowledge gaps. Table 1 illustrated the topic coverage in Python and that each week, students are assigned reading that covers those chapters and must complete the reading quizzes for participation credit. The course also utilizes some of the *code checks* from the text for participation credit and in-class exercises.

There are several new concepts for those students who have only been exposed to Java, such as functions, tuples, lambda expressions, and list comprehensions. The syntax for a lambda function is introduced in the context of sorting lists of tuples where the sort requires an order that is not the default. List comprehensions are an elegant short-cut syntax for creating lists.

The Python programming assignment reads in a csv file of clean data and processes the data into objects, using concepts from object-oriented database design [6]. In a weather example with stations reporting observations, the students create Station and Observation classes and the list of observations for a station is a list of references to its associated observation object instances. The students are required to create a dictionary of these objects to essentially store the "database extent". Using this data structure, students answer specific questions by coding the navigation of the data and perhaps creating additional data structures in the process. Due to the increased enrollment of the class

(now capped at 60), a free replit classroom [14] for the Python language assists students with checking their output with the expected output. The programs are still assessed based on a visual inspection of the code with a detailed rubric because the assignment description provides strict guidelines on the format of the objects and data structures to be incorporated.

5 Erlang

Erlang is a dynamically typed, functional language that has a lexical structure similar to Prolog for atoms, variables, and list processing. Erlang includes maps, tuples, lists, and strings, as well as lambda expressions and list comprehensions. Thus, Erlang is chosen as the next language in the sequence, building on these similar concepts from Python as well as functions. Erlang only supports recursion and not iteration, so the list processing is similar to Prolog with a head and a tail, however, the Erlang rules are functional and utilize pattern matching. The first rule head that matches is chosen, and variables are bound once through the pattern matching. The coverage of Erlang is restricted to sequential Erlang since the only prerequisite is CS2 and not operating systems. Students are directed to both a free online book [9] and the Erlang site [7] for online documentation.

There are many new concepts for students to grasp with Erlang, including its functional, pattern matching approach as well as the use of recursion instead of iteration. The processing of lists using recursion with the head and tail construct is a skill that students must learn to master with Erlang.

The Erlang programming assignment is similar to Python with the same data but a value-based relational database approach stores the relationships between the data as id values as components of tuples, which are stored in maps. In the weather example, there are `station` and `observation` tuples that are associated by the station's unique code. To simplify the assignment, students are provided with a template for the program that handles the reading of a csv file, returning a list of strings for each line. They must complete the program to create the maps and answer the questions over the data. Students use an online Erlang replit environment to complete their assignment because an Erlang classroom was not available as of Spring 2020. Therefore, students are given a compiled Erlang file and they must call a function in this module to validate their data structures before submission.

6 Prolog

The fundamentals of Prolog as a logic language are covered, including the details of clauses, resolution, and a refutation proof so students understand

that Prolog provides binding to variables when it has found a refutation and backtracking continues searching when asked for the next answer. However, the emphasis on Prolog is quickly switched to using Prolog facts to store data and its declarative rules to compute the answers to database queries. This is a practical approach that students appreciate. Replit does not support Prolog. However, SWI-Prolog has an online interpreter [20]. There is an online resource for Prolog [2] and our students have online access to [3] through the university library.

The codingbat and review exercises in Prolog are quite similar to that of Erlang with a fundamental difference. While both Erlang and Prolog must pass the information that it needs, Erlang functions return values but Prolog must return values by adding an extra argument to bind the result to a variable. Since the class does not cover extensions to fundamental Prolog, the review exercise question that creates a letter-count map is changed to the creation of a list of letter(Letter, LetterCount) tuples for each letter in the string.

The Prolog programming assignment uses the same set of data, now stored as Prolog facts. For the weather example, the facts are stored as station and observation tuples, with relationships using the value-based ids. The assignment asks students to answer queries over that data in Prolog. There are five queries that typically involve returning a list of tuples sorted in a particular manner. Since asking a database query typically means to find all answers, the Prolog predicate findall(Template, Goal, Bag) is introduced, which returns a list of bindings for the Template satisfying the Goal in the Bag variable. Similar to a list comprehension, the Template represents the term for inclusion in the list where the bindings are satisfied by the Goal.

7 Reflection

The last week of the semester, while students are working on their Prolog assignment, the class begins a reflection phase, looking back on what was learned at the same time that higher-order programming is introduced. Essentially, higher-order programming is where code can be treated as a value, passed into or returned from a block of code. The lambda functions that students used in Python and Erlang as sort functions are examples of higher-order programming. The course notes emphasize the filter and map higher-order functions available in the languages, although in Prolog the predicates are called include and maplist. To show the importance of handling data as streams of information, an overview of Java streams is covered to illustrate the filter and map in Java.

As the class reviews context-free grammars for the comprehensive final, class time is allocated to show students Prolog's support of definite clause

grammars. In fact, SWI-Prolog provides a grammar example to run [19]. In addition to this demonstration, an example grammar from earlier in the semester is modified to a definite clause grammar and the parse trees that were hand-drawn at that time are visually represented by Prolog.

Learning three new programming languages is a challenge for most students. They are learning new ways of thinking about problem solving. Erlang and Prolog's reliance on recursion instead of iteration is particularly challenging for students who have only completed CS2 and have not taken the full Data Structures and Algorithms class. There are a significant number of students who rise to the challenge and embrace the new paradigms. Although anecdotal, the exposure to multiple programming paradigms, and in particular Prolog, has helped students in learning SQL in the required database class. Also, the exposure to a functional language helps students learn LINQ (Language INtegrated Query) [13] in the advanced database course.

References

[1] T. Austin, C. Horstmann, and H. Vu. Explicit short program practice in a programming languages course. *Journal of Computing Sciences in Colleges*, 33(4):114–122, 2018.

[2] P Blackburn, J Bos, and K Striegnitz. Learn prolog now. `http://www.learnprolognow.org/`.

[3] M Bramer. *Logic programming with Prolog*, volume 9. Springer, 2005.

[4] Y Chen and W Tsai. *Introduction to programming languages: Programming in C, C++, Scheme, Prolog, C#, and SOA, 5th ed.* Kendall Hunt Publishing Company, 2017.

[5] W R Cook. High-level problems in teaching undergraduate programming languages. *ACM Sigplan Notices*, 43(11):55–58, 2008.

[6] S W Dietrich and S D Urban. Fundamentals of object databases: Object-oriented and object-relational design. *Synthesis lectures on data management*, 2(1):1–173, 2010.

[7] Erlang. Erlang programming language. `https://www.erlang.org/`.

[8] M Gabbrielli and S Martini. *Programming languages: principles and paradigms*. Springer Science & Business Media, 2010.

[9] F Hebert. Learn you some erlang for great good! `https://learnyousomeerlang.com/`.

[10] C S Horstmann and R D Necaise. *Python for everyone, 3rd ed.* Wiley Publishing, 2018.

[11] K D Lee. A framework for teaching programming languages. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 162–167, 2015.

[12] M C Lovett. Make exams worth more than the grade: Using exam wrappers to promote metacognition. *Using reflection and metacognition to improve student learning: Across the disciplines, across the academy*, pages 18–52, 2013.

[13] E Meijer. The world according to linq. *Commun. ACM*, 54(10):45–51, October 2011.

[14] Repl.it. Repl.it - the collaborative browser based ide. `https://repl.it/`.

[15] M Sahami, A Danyluk, S Fincher, K Fisher, D Grossman, E Hawthorne, R Katz, R LeBlanc, D Reed, S Roach, et al. Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science. *Association for Computing Machinery (ACM)-IEEE Computer Society*, 2013.

[16] S Samuel. Teaching programming subjects with emphasis on programming paradigms. In *2014 International Conference on Advances in Education Technology (ICAET-14)*. Atlantis Press, 2014.

[17] M L Scott. *Programming Language Pragmatics, 4th ed.* 20016.

[18] R W Sebesta. *Concepts of programming languages, 11th ed.* Pearson Education, Inc, 2016.

[19] SWI-Prolog. Swish – grammar example. `https://swish.swi-prolog.org/example/grammar.pl`.

[20] SWI-Prolog. Swish – swi-prolog. `https://swish.swi-prolog.org/example/examples.swinb`.

[21] E Tilevich, C A Shaffer, and A C Bart. Creating stimulating, relevant, and manageable introductory computer science projects that utilize real-time, large, web-based datasets. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 711–711, 2015.

[22] J Traxler. Choosing languages to teach programming paradigms and programming styles. *WIT Transactions on Information and Communication Technologies*, 7, 1994.