4-23-2019

Teaching Fluency in German and Java: A Comparison of Instructional Methods for Foreign Languages and Programming Languages

Jennifer L. Berry

Teaching Fluency in German and Java:
A Comparison of Instructional Methods for Foreign Languages and Programming Languages

By

Jennifer L. Berry

An Honors Capstone
submitted in partial fulfillment of the requirements
for the Honors Diploma
to

The Honors College

of

The University of Alabama in Huntsville

April 23, 2019

Honors Capstone Director: Dr. Richard Coleman
Faculty Lecturer of Computer Science

Jennifer L. Berry 4-24-19
Student Date

[illegible] 4-23-2019
Director Date

[illegible] 4-23-2019
Department Chair Date

[illegible] 5/1/19
Honors College Dean Date

HONORS COLLEGE
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Honors College
Frank Franz Hall
+1 (256) 824-6450 (voice)
+1 (256) 824-7339 (fax)
honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted as a bound part of the thesis.
In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Jennifer L. Berry
Student Name (printed)

[illegible]
Student Signature

4-24-19
Date

Teaching Fluency in German and Java: A Comparison of Instructional Methods for Foreign Languages and Programming Languages

By

Jennifer L. Berry

An Honors Capstone
submitted in partial fulfillment of the requirements
for the Honors Diploma
to

The Honors College

of

The University of Alabama in Huntsville

April 23, 2019

Honors Capstone Director: Dr. Richard Coleman
Faculty Lecturer of Computer Science

Student	Date
Director	Date
Department Chair	Date
Honors College Dean	Date

HONORS COLLEGE
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Honors College
Frank Franz Hall
+1 (256) 824-6450 (voice)
+1 (256) 824-7339 (fax)
honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted as a bound part of the thesis.

____
Student Name (printed)

____
Student Signature

____
Date

Dedication:

Special thanks to Melanie Berry, for being my “secret weapon” in all major writing endeavors, including this one.

Many thanks also to my family and to John, for encouraging me throughout my college career.

Table of Contents

Abstract 2

Introduction 3

Topics Not Covered in this Paper 5

Issues Addressed by this Paper 5

The Basis of Programming Languages is Natural Language 9

Historical Basis 9

Language Classification of Programming Languages 10

Structural Basis 11

Current teaching techniques for programming and foreign languages 15

Grammar Teaching Separated from Programming Language Teaching....................15

Lack of Application Involved in Practice Programming Exercises 18

Context Missing from Programming Language Teaching 20

Insufficient Feedback on Programming Practice 24

Proposed Solutions based on Foreign Language Pedagogy 26

Explain Programming in an Active and Engaging Manner....................26

Show How Programming is Useful....................29

Give Students Agency in Selecting a Programming Language 32

Provide Rapid Feedback....................34

Concluding Remarks....................37

Works Cited....................39

Abstract

Students in introductory programming courses often find that learning a programming language is a difficult endeavor. Introductory programming classes are taught much differently than foreign language, despite commonalities between programming languages and foreign languages. This paper identifies ways in which foreign languages and programming languages can be considered pedagogically similar. Current issues related to programming pedagogy are addressed, and solutions are provided by examining successful aspects of foreign language pedagogy. Ultimately, this paper seeks to provide a new, creative perspective on teaching programming that will appeal to a larger body of students. Pedagogical techniques for programming based on how foreign languages are taught can attract creative students who will further computer science as a discipline with innovative ideas.

Introduction

"It's like learning a foreign language!" is a common consensus of students learning introductory C programming at the University of Alabama in Huntsville (UAH). Students who are struggling with syntax, unsure how to construct a program, or plagued with cryptic error messages find themselves lost in terminology, doubtful that programming is even worth the effort. As a peer tutor at the UAH Student Success Center, I was surprised to find that students taking introductory C programming were the largest student group seeking my assistance. Instead of helping students with the notoriously difficult Discrete Mathematics course, or the fundamental Data Structures course, or even the detail-oriented Assembly Language course, a majority of the students seeking my help were not struggling with concepts I expected them to find challenging. Instead, learning a programming language for the first time is extremely difficult for students in introductory programming classes.

I was puzzled by the discovery that many students struggle with introductory programming courses. I originally thought that so many students from introductory programming came to see me in the tutoring center because more students are required to take introductory programming; at UAH, all College of Science Majors have to take this course. However, as I continued to see large volumes of students from this course each semester, I realized the larger volume of students was not the most likely cause for the great need of tutoring assistance. The types of problems students of introductory C programming bring to tutoring sessions stem from a deeper issue with the way that programming is being taught to them.

Students often come to tutoring clueless about how to study for programming tests, unsure where to begin writing programs, and unable to resolve debugging errors from their programs. What makes programming different from other classes students are taking, to the point where they are not sure how to approach the subject? Why is composing programs so difficult for beginning students? If students need creative problem-solving for program solutions and creative ways to study programming language concepts, why is there no creative emphasis on programming? If the medium used to program is called a programming language, why is this subject not treated like learning a new language?

Students of foreign language can find themselves lost in grammar rules and feeling overwhelmed by the cultural contexts in an introductory class, much like students of programming. Second language acquisition is difficult, if for no other reason that the 'foreignness' of the language; it is similar in some aspects to the student's native language, but it differs in many ways from the lexical rules and phoneme[1] meanings the student has learned from infancy. How, then, do students of a foreign language not only learn new grammar rules and vocabulary terms, but also learn how to apply those rules in normal conversation or composition in a new language? What are foreign language classes doing to facilitate this sort of application of a new language that programming language classes are not? If foreign language pedagogy functions better for students and has the components that introductory programming courses need, why is programming pedagogy not borrowing techniques from foreign language? If introductory courses are trying to teach a new language to students, why are they not taught more like foreign language classes? This paper seeks to bring a new perspective and a creative lens to the pedagogy of programming languages by treating them as foreign language for students to learn and use.

[1]A word, sound, or syllable – the smallest component of a language

Topics Not Covered in this Paper

Because foreign languages can be learned in widely varying contexts, it is useful to define the specific language learning settings examined by this paper. This research is not focused on early language acquisition, such as a toddler learning his or her native tongue; although psychologically fascinating, this is not as similar to the situation encountered by a student learning to program in college because a college student already knows how to use a natural language. This paper will also not be examining the similarities of the psychological process of learning a foreign language or a programming language; that comparison requires far more research and experimentation that do not further the main purpose of this paper.

It is also important to note that some parts of the language learning process are not comparable to learning a programming language; namely, learning a new alphabet or symbol system, and learning how to pronounce new phonemes in the target language. Although these are important parts of the language learning process, they are irrelevant when considering programming language learning, which requires typing using a familiar alphabet and does not have an auditory component. Because these elements are not easily related to programming languages or their pedagogy, they are not relevant to the object of this paper.

Issues Addressed by this Paper

While numerous sources discuss various educational techniques and explore the relationship between faculty and students in higher-level education, only a small subset is focused on teaching programming; the ACM[2], in a bulletin gathering literature useful to computer science professors, confirms that "there is a large body of general literature on the roles and methods teachers and students encounter in higher education…[however,] while it is

[2] Association for Computing Machinery

important for computer science teachers to be aware of this literature, much of it is not directly relevant to teaching introductory programming"[3]. Most of the papers used to explain programming pedagogy are "evidence-based … defined as the use of literature and results from relevant areas of the scholarship of teaching and learning to motivate and support teaching practice" (204). Pedagogical literature that directly addresses programming, because it is often evidence-based, typically presents a pedagogical idea with a specific example in mind; the environment, programming language, or course methodology are elements of a research experiment, and the literature is assessing how well these techniques worked. The environment and language in which students learn to program, then, plays a large role in pedagogical literature focused on teaching programming.

By contrast, foreign language pedagogy is often considered from a more general perspective, with examples from specific languages used to exemplify the pedagogical concept being shared. Computer Programming, then, is often too focused on the success of teaching the syntax and structure of one specialized language or virtual environment[4] than it is on teaching students how to program without regard for language or environment. This paper is seeking to

---

[3] Pears, Arnold, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. "A Survey of Literature on the Teaching of Introductory Programming." *ACM SIGCSE Bulletin*39, no. 4 (2007): 204-23. doi:10.1145/1345375.1345441, 206.

[4] Such as, for example, the Virtual Programming Lab studied in: Cardoso, Marilio, Antonio Vieira De Castro, and Alvaro Rocha. "Integration of Virtual Programming Lab in a Process of Teaching Programming EduScrum Based." 2018 13th Iberian Conference on Information Systems and Technologies (CISTI), 2018. doi:10.23919/cisti.2018.8399261.

expand the way that programming is considered as a subject, not just as a set of specialized skills segmented by programming language.

Of course, several differences exist between programming and speaking a foreign language. The parts of programming language that remain, however, are very similar to how natural languages function. This research begins by examining the similarities between programming languages and natural languages in order to justify their comparison as pedagogically similar.

In order to address the problems that exist in introductory programming language classes, this research identifies what the root issues are. Once these have been identified, this paper provides improvements that could solve these fundamental problems, based on the elements that make foreign language classes successful. Poor curricula "complicates the absorption process for these subjects, negatively affects the student study-rate achievements, and also reduces the popularity of IT studies in general"[5]; improved methods for teaching programming are required to attract more students to the rewarding and rapidly growing computer science discipline.

Ultimately, if introductory programming classes continue in the way they are being taught now, few students will learn programming as a creative practice that is valuable to their academic or professional lives. Instead, programming will only be attempted by those who enjoy logic and mathematics, and not by those who want to solve problems creatively. Because professional programmers must be creative problem solvers who apply fundamental software engineering concepts in large, complex software products, programming should be taught as an

[5]Daiva Vitkute-Adzgauskiene and Antanas Vidziunas, *Problems in Choosing Tools and Methods for Teaching Programming*, vol. 11 (Kaunas, Lithuania: Informatics in Education, 2012), PDF, 280.

inventive subject. If the creative application of programming is not taught to introductory students, Computer Science will lose a body of students who could further the discipline in innovative and exciting new applications. The solution to problems plaguing introductory programming classes can be found by considering programming as a creative and analytical subject, like foreign language.

The Basis of Programming Languages is Natural Language

Historical Basis

Programming languages came about because humans needed a better way to interface with computers. To a computer, all data and instructions related to that data must be given in sequences of 1's and 0's; these binary strings hold special meaning to the computer about what operations it should do to what data, and are sequenced correctly so the computer completes these operations in order. To humans, however, strings of 1's and 0's do not hold any special meaning. It is very tedious for humans to parse through what these binary numbers mean, and even more tedious to specify binary strings correctly for a program.

Programmers in the 1960s decided there had to be a less error-prone way to specify instructions to the computer[6]. They developed a shorthand language that was more easily readable by humans, but was easily translated by the computer into the 1's and 0's it needed. This solution was called assembly language and is still used (if somewhat infrequently) to write programs today[7]. If computers could translate human readable codes into binary strings, though, why not make those codes even more intuitive for the humans writing them? Shortly after assembly language was developed, early programmers developed more complex codes, which, while still ultimately translatable by the computer into the binary strings it needed, looked like a

[6] Scott, Programming Language Pragmatics, 2004, 2.

[7] Pal, Kaushik. "Why Is Learning Assembly Language Still Important?" Techopedia.com. Accessed April 22, 2019. https://www.techopedia.com/why-is-learning-assembly-language-still-important/7/32268.

human language[8]. This allowed programmers to use a language, based on their own native language, which allowed them to more easily interface with a computer.

Of course, some elements of programming languages were simpler than programmers' native languages; the computer does not talk back, and there can be no shades of meaning in code. The language used to interface with the computer, however, was still fundamentally based on the way humans communicate, because it was written by humans to facilitate communication with a computer. Because modern code is still interpreted by the computer into machine-level instructions, the computer does not care whether its instructions were originally written in any particular code; the strings of 1's and 0's are roughly the same, and the resulting program is the same from the perspective of the computer. Programming languages were made for human use, and based on human language, so they ought to be taught as what they are: a variant of natural language.

Language Classification of Programming Languages

Because of their design and purpose, programming languages easily fit the definition of a language. Language can be defined or described as "a system of meaning… by which meaning is created and meanings are exchanged", exemplified either by two people speaking to each other to express ideas, or by a programmer specifying a line of code which holds some underlying meaning (a corresponding set of binary strings) for the computer[9]. Language can simplistically be reduced to the set of combinations of the smallest units of meaning (signs, words, utterances,

---

[8] Scott, Michael L. *Programming Language Pragmatics*. San Francisco, CA: Morgan Kaufmann, 2004.

[9] Halliday, Michael Alexander Kirkwood, and Jonathan James Webster. *On Language and Linguistics*. London: Continuum, 2004, 2.

or tokens[10]) to produce some overall meaning that can be interpreted by a second party (another person, a group of people, or even a computer). Another definition describes language as "a symbolic communication system, or in one word … a 'code'"[11].

All of the definitions given here are used to describe natural languages; they are also easily descriptive of programming languages because of how a programming language is used to specify meaning that both a programmer and a computer can understand[12]. Because programming languages fit the definition of a language, it is reasonable to draw parallels between natural and programming language usage and pedagogy. If programming language is taught in such a way that its origins in natural language, its grammatical structure, and its definitive linguistic elements are overlooked, Students are not taught the underlying purpose, function, and application of programming.

Structural Basis

Because programming languages are based on natural language, they share several common traits. One very obviously common trait is the usage of grammar to define permissible combinations of words and symbols to create some meaning. A language, whether natural language or programming language, is simply a set of strings of characters of some alphabet; the syntax rules of a language specify which combinations are legal and meaningful to the

[10] The smallest units of a language: words and punctuation both count as tokens. In computer science, a token could be a reserved word, variable name, symbol or operator.

[11]Wilga M. Rivers, *The Psychologist and the Foreign-Language Teacher* (Chicago: Univ. of Chicago Pr., 1976), 23.

[12] Scott, Programming Language Pragmatics, 2004, 2.

language[13]. While syntax rules exist for both natural languages and programming languages, they are much simpler and easier to explain in programming languages than in a natural language (73). Despite its relative simplicity when compared to natural language grammar, the syntax of a programming language has the same function as the grammar of a natural language; it defines the set of rules for combining the parts of the language in the proper order to create some meaning.

The similarities of programming language and natural language grammars are evidenced even by how they are visually depicted. The Backus-Naur form of specifying a programming language's grammar rules is directly based on the definition of Context-Free grammars, explained by linguist Noam Chomsky (75). Chomsky's research on context-free grammars emerged just before the release of the programming language ALGOL[14], for which the Backus-Naur form was developed originally (75). Expressing a programming language grammar in the same way as a natural language grammar became extremely popular because it was a concise way to express how sentences (or statements of code) could be concisely and clearly expressed (75). Both programming language and natural language grammars are depicted as being parsed in this way; the meaning of a sentence can be broken apart into component meanings (the meanings of individual words) and re-assembled into the contextual meaning of the whole sentence (or statement of code). The direct similarities between Chomsky's grammar parse trees

[13] Sebesta, Robert W. *Concepts of Programming Languages*. Ed Wood City, Cal.: Benjamin/Gumming, 1993, 73.

[14] Algorithmic Language; developed in the 1950's to overcome several perceived problems with the programming language FORTRAN. Algol was influential to later languages, like Pascal and C.

and the Backus-Naur parse trees showing how languages are parsed are apparent in Figures 1 and 2.

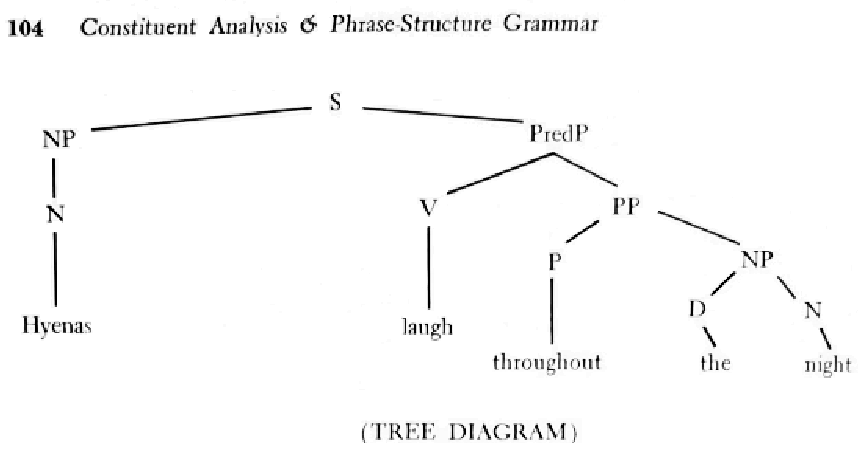

Figure 1: a parse tree showing how a sentence (in English) might be parsed using a parse tree; the sentence is "Hyenas laugh throughout the night"[15].

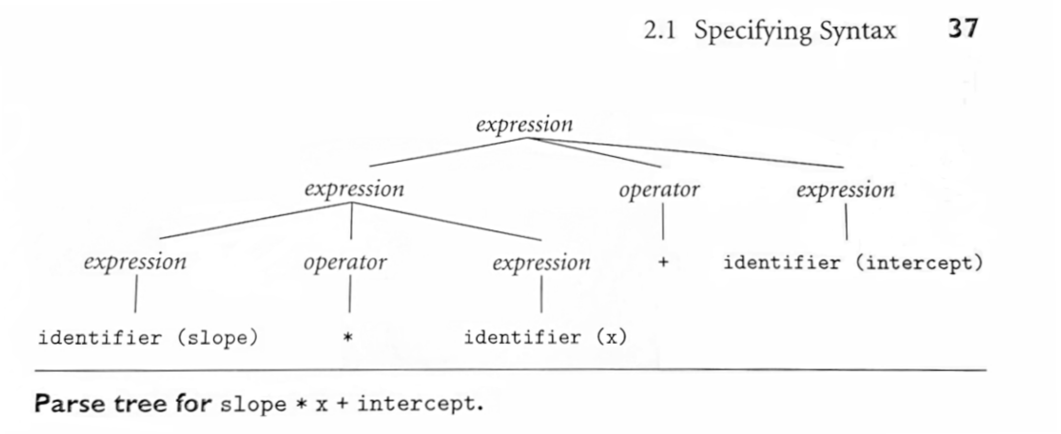

Figure 2: a parse tree showing how a computer can parse the grammar of an assignment statement, `slope * x + intercept`.[16]

[15] Beechhold, Henry F., and John L. Behling. *The Science of Language and the Art of Teaching*. New York: Scribner, 1972, 104.

[16] Scott, *Programming Language Pragmatics*, 2004, 37.

The purpose of these parse trees, both for programming language and foreign language, is to show how a statement of meaning can be broken down into component parts and re-assembled to construct the entire statement. Parse trees, then, are describing how humans (or computers) can understand language by breaking it apart and re-assembling it to express a thought or action using small words or tokens. The process used by the computer to parse statements of code can be described the same way as the process used by a human to parse meaning from a sentence. Not only are programming languages and foreign languages structured in a similar manner, but their structure is used to produce meaning in a similar manner.

Current teaching techniques for programming and foreign languages

Grammar Teaching Separated from Programming Language Teaching

Because of their common basis, foreign languages and programming languages have a similar underlying structure. Grammar can be a complicated topic to teach beginners. To foreign language students, "the word 'grammar' brings to [mind]… a formal and often uninteresting analysis of language…conjugations, paradigms, declensions, and diagraming, all … appear to be an end in themselves"[17]. To programming students, grammar is referred to as syntax, which can seem like arbitrary, strict rules about how to write a program. The fundamental structure for both natural languages and programming languages, although fundamental to correct usage of the language, is often taught in a burdensome, dry manner. Grammar teaching can be a problem for both foreign language and programming languages; the difference between them is that foreign language pedagogy recognizes the problems associated with grammar teaching and proposes solutions.

Programming courses have historically aimed at four categories of skills, taught sequentially: using the programming environment; understanding syntactic rules; analyzing code written by others and understanding its semantics[18]; and generating their own programs[19]. If

[17] Allen, Edward David., and Rebecca M. Valette. *Classroom Techniques: Foreign Languages and English as Second Language*. Harcourt Brace Jovanovich, 1977, 81-82.

[18] What the program does: its meaning with respect to what happens when the program is run. This can also be referred to as 'axiomatic semantics': Scott, *Programming Language Pragmatics*, 2004, 170.

syntax is taught second, it comes before the student can fully understand what code is doing. Separation of syntax and semantics leaves students confused about the necessity of particular rules. Beginning programmers can struggle to understand how the syntax relates to what the program does; "one study of novice programmers showed that … substantial misunderstandings had occurred with regard to virtually every construct in the language"[20].

Syntax is crucial to program writing; if the syntax is wrong, the computer cannot understand the program, and it will not compile or run. Programs that do not compile or run are very difficult for beginners to troubleshoot, causing frustration for novice programmers. Difficulty applying syntax rules is not a new problem for beginners, and "years of research have shown that students have several difficulties in learning how to program. The major difficulty experienced by novices is how to combine and properly use the basic structures to build a program"[21]. Syntax is simpler than natural language grammar and should therefore be relatively easy to explain. The application of this syntax, however, is lost for beginning students (169). This leads to broken programs and students without a clue why their code does not work, because they do not understand the syntax well enough to identify what should be a simple

[19] Van Merrienboer, Jeroen J. G., and Hein P. M. Krammer. "Instructional Strategies and Tactics for the Design of Introductory Computer Programming Courses in High School." *Instructional Science*16 (1987), 253.

[20] Sleeman, Derek. "The Challenges of Teaching Computer Programming." *Communications of the ACM*29, no. 9 (1986), 840.

[21] Aureliano, Viviane Cristina Oliveira. "A Methodology for Teaching Programming for Beginners." Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research - ICER 13, 2013. doi:10.1145/2493394.2493417, 169.

problem. Syntax cannot be merely presented to students as a set of rules to memorize; it has to be integrated into an explanation of what the code structure does, so that students understand how to apply it to create programs that run.

Foreign language teaching techniques integrate language practice into grammar lessons. To keep grammar from being overwhelming or boring, it must be presented as part of expressing meaning in the language in order to promote student interest and involvement. When grammar is taught only as a set of rules to be followed, language learning becomes mere memorization and rule-following, which can seem overwhelming or uninteresting for the beginning student. Even some modern techniques for teaching grammar are not always a significant improvement, because they still comprise of "rules, definitions, memory work, and busy work"[22]. Grammar can seem less of a burden, however, if "students are eager to communicate their thoughts, and if to do so they must select the proper forms and put them in the correct order"[23]. Grammar must be presented as directly applicable and useful to students seeking to express their ideas in a new language. In order to be successful, grammar study must be "'organic' (a part of the larger study of language in all its diversity and applications) and exploratory (i.e., examined via the inquiry-discovery method)"; otherwise, the topic of grammar might as well be abandoned entirely[24].

What foreign language takes into account about grammar teaching that computer science misses about syntax teaching is the application component. Students of foreign language understand that the meaning of their sentence changes if the wrong grammar is used; they realize

[22] Beechhold and Behling, *The Science of Language and the Art of Teaching*, 1972, 129.

[23] Allen and Valette, *Classroom Techniques: Foreign Languages and English as Second Language*, 1977, 81-82.

[24] Beechhold and Behling, *The Science of Language and the Art of Teaching*, 1972, 129.

that if they use the wrong grammar, they will not be understood and will be severely limited in language expression. Foreign language students, then, are motivated to keep practicing grammar until they are correctly understood, and are thus able to learn how to express themselves appropriately without becoming frustrated. Students of programming, however, see syntax as a hurdle or challenge of programming that requires memorization of rules. Code syntax appears strict and arbitrary, and is not seen as the way the computer can understand instructions. Programming language syntax, then, should be treated more like a language grammar and not like mere rules or formulas that must be memorized.

Lack of Application Involved in Practice Programming Exercises

In an example foreign language curriculum, a set of videos with a storyline immerse the student in the target language, and after watching and discussing the video a few times in a row, the students move from copying the dialogue presented in a film, to making changes in the dialog to reflect their own name or to answer other questions about the story[25]. The film is not just a set of arbitrary words or phrases; it has some substance to it and is more natural and useful to students than a set of arbitrary example sentences. The application of the new language to tell a story, whether the story in the video or their own, helps students use the target language more effectively and to see its applications more clearly than practice of words or phrases outside of a story (27).

Students may perform other many exercises called structure drills; in the drill, "each change in form has demonstrable consequences in terms of demonstrable meaning", and changes in the grammar or individual words of a sentence are demonstrated with a change in the image or

[25] Oller, John W., Jr., and Patricia A. Richard-Amato. *Methods That Work: A Smorgasbord of Ideas for Language Teachers*. Taipei: Wen He, 1984, 28.

a depiction of some different action to help the students link the changes in the phrase to how they affect the meaning (28). As students continue into more advanced courses, the goal shifts from just learning grammar rules to acquiring them; namely, that students would internalize the grammar rules and feel more natural using them, rather than just simply reproducing of these rules (31). Thus, students move from simply remembering the rules and vocabulary to even more extensive application as they develop their language skills. In all aspects of language practice, students are given active and engaging ways to apply the rules they learn to express new ideas, provided with concrete referents or experiences that connect to the meaning of new words or phrases, and shown how to apply what they are learning in foreign language class to their lives. Their practice is not simply chosen to strengthen their language skills, but also to develop their understanding of the culture and critical thinking skills.

Students in programming classes must also sense how the concepts they are learning apply to their lives, or they will not be interested in practicing and learning more about programming. Teachers of introductory classes are beginning to realize that they must "introduce language features along with coding practice to demonstrate and enhance understanding of language syntax and sentence structures"[26]. Teaching language features and syntax is useless without engaging examples that students will view as useful or applicable. Part of the fun of programming is solving a worthwhile problem; if students do not feel that the problems they are solving are useful, they will not view programming as enjoyable or applicable to their lives. For students who want to pursue programming professionally, "[their] career plans are affected by

[26] Suo, Xiaoyuan. "Toward More Effective Strategies in Teaching Programming for Novice Students." *Proceedings of IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE) 2012*, 2012. doi:10.1109/tale.2012.6360379, 1.

the curricula structure, IT needs in the market, university teaching and research labs activities, and overall scientific orientation of the university”[27]. Student interest, then, is shaped by the quality of the curriculum and the applicability of their practice, whether the students are computer science majors or not.

Context Missing from Programming Language Teaching

Students of foreign languages select their target language based on their interest in the culture[28]. Some teachers of foreign language accept exposure to culture as a sufficient learning outcome of the foreign language curriculum; even if the student has very limited proficiency in the language, at the very least they will have gained some new understanding of another culture as well as a different perspective of the world[29]. A foreign language course without any discussion of culture or history would be counterproductive. The new perspective of the world gained through learning a foreign language is crucial and is a worthwhile objective of language courses[30].

---

[27] Vitkute-Adzgauskiene and Vidziunas, *Problems in Choosing Tools and Methods for Teaching Programming*, 2012, 279.

[28] Among other factors, like relatives who speak the language or perceived value of language as a professional skill, and so on: Adams, Heather. "How Students Choose Their Foreign Language Classes." KBIA. April 29, 2014. Accessed April 22, 2019. https://www.kbia.org/post/how-students-choose-their-foreign-language-classes#stream/0.

[29] Oller and Richard-Amato, Methods That Work: A Smorgasbord of Ideas for Language Teachers, 1984, 3.

[30] Beechhold and Behling, The Science of Language and the Art of Teaching, 1972, 17.

Programming languages also have their own secondary objective; they are “usually expected to foster the development of specific cognitive skills which may positively affect problem solving behavior in other school disciplines”[31]. Problem solving skills could assuredly be useful for students regardless of whether they write programs professionally or not. Programming courses, however, often fail to meet this objective; “reports from teachers of programming and results from some empirical studies now suggest that the teaching of programming has created significant difficulties for high-school and university students, and has failed to catalyze the development of higher order thinking skills”[32]. Programming courses fail to meet their objective of problem-solving skills because they hope that problem solving skills will arise as the students learn to program, rather than teaching problem solving techniques to help students learn programming[33].

Problem-solving skills are a worthwhile objective of introductory programming courses, just like exposure of culture is a worthwhile outcome of foreign language courses. To achieve this outcome, foreign language courses integrate culture studies into language studies through discussion and language practice. Programming language courses, by contrast, hope to achieve their objective simply by teaching programming. In reality, problem solving skills facilitate

---

[31] Van Merrienboer and Krammer, "Instructional Strategies and Tactics for the Design of Introductory Computer Programming Courses in High School", 1987, 252.

[32] Sleeman, "The Challenges of Teaching Computer Programming", 1986, 840.

[33] Palumbo, David B. "Programming Language/Problem-Solving Research: A Review of Relevant Issues." *Review of Educational Research* 60, no. 1 (1990): 65-89. doi:10.2307/1170225, 67.

learning to program, and developing problem solving skills organically is often very difficult for beginning programmers[34]. Teachers of language courses understand that students can more easily remember vocabulary and grammatical structure if they understand the culture behind it. Teachers of programming courses, unfortunately, do not have the same insight. The programming pedagogy handles problem solving skills in programming courses is somewhat backwards from what students need.

One aspect of problem solving that students struggle with when trying to compose programs independently is the breakdown of larger problems into smaller ones. Foreign language students can do this easily, breaking down pages into paragraphs, paragraphs into sentences, and so on. Composition of ideas functions much the same way as it does in students' native languages. Programming students, by contrast, cannot infer how they should proceed in completing assignments. Beginning programming assignments are often given to students as one problem, and breakdown is expected to occur naturally (23-24). This, however, is an overwhelming task for novice programmers; while problem decomposition seems intuitive to expert programmers, beginners have difficulty deciding how they should divide a large problem into smaller ones[35].

Foreign language students can compose their thoughts into sentences and paragraphs in the target language because they are simply applying their knowledge of composition techniques in their native language with a new language's grammar and vocabulary. Because programming

[34] Hancock, Chris. "Context and Creation in the Learning of Computer Programming." *For the Learning of Mathematics*8, no. 2 (February 1988): 18-24. Accessed January 10, 2019. https://www.jstor.org/stable/40248137, 18.

[35]Sleeman, "The Challenges of Teaching Computer Programming", 1986, 840.

students are not considering programming as a language, they do not think to apply outlining techniques to algorithm design. Lack of forethought and neglect to implement familiar outlining techniques to set up the structure of their program causes "students [to] implement the entire program with little or no design"[36].

While algorithm composition can seem mathematical at first glance, this process also shares many similarities with composition in language. Programming students need not learn algorithm composition with no prior knowledge; they must only learn a new way of expressing themselves to a new audience. Research has already established that "teaching programming design to assist problem solving has been widely recognized as an effective approach in traditional programming classes"; if programming teachers can connect programming design to what students already know about writing, programming teachers can build off of their prior knowledge of language instead of starting from scratch (2). Application of these already familiar skills could simplify programming significantly for beginning students.

If students write programs without first designing them, "[they] will have to deal with quantities of errors at a time" and become very frustrated; "debugging is a negative experience" for novices attempting to complete programs in this manner (2). Like the writing student without an outline, a programming student without an algorithm who simply starts to code will end up with an inefficient program that is hard to read and debug.[37] Frustrating troubleshooting and

[36] Suo, "Toward More Effective Strategies in Teaching Programming for Novice Students", 2012, 2.

[37] Such a program is typically referred to as "spaghetti code" because it it tangled, messy, and disorganized.

inability to break down a large problem into component parts is the result of students not approaching programming language like they approach other language classes.

Insufficient Feedback on Programming Practice

When students practice speaking in a foreign language class, the teacher can immediately give feedback, whether affirming that the student has formed the sentence correctly, suggesting different vocabulary, or correcting grammatical elements of what the student has said. The student knows almost instantaneously if they are understanding the concept. Quick feedback prevents students from making the same mistake for a long period of time, thus making mistakes easier to correct, while also allowing students to see their own improvement in the target language.

In programming classes, the student will immediately be able to tell if their program is incorrect; if it does not compile, if the output is wrong, or if the program encounters an error during execution, something about their program is obviously wrong. The trouble is that beginning students cannot understand the error messages generated by their programming environment, and they have great difficulty discerning why their output is wrong. The feedback given by the computer is not sufficient feedback for the student, because the student cannot discern the necessary correction. The teacher must help the student one-on-one to interpret the computer's feedback for the student about what is causing the problem, and what they must do for their program. One-on-one work and grading of programs are tedious, and takes teachers a substantial amount of time; the result of this is that "feedback to students about their own assignments is neither complete nor given quickly"[38]. When this instructor feedback is delayed

[38] Cardoso, De Castro, and Rocha, "Integration of Virtual Programming Lab in a Process of Teaching Programming EduScrum Based", 2018, 2.

absent, “students cannot develop their work and improve their programming skills as fast as desirable and they [are not] quite sure if what they did is correct” (2).

At the University of Alabama in Huntsville, usage of built-in debuggers and simple strategies for testing programs, like “echo debugging”[39] are not explicitly taught to students until their third or fourth programming course, if at all[40]. Students seeking tutoring help can typically understand how to use simple debugging techniques from only a brief explanation[41], and these troubleshooting techniques are indispensable for the student when trying to figure out what went wrong and why. Debugging tools are useful for programmers, from absolute beginners to professionals – and yet, even simple echo debugging and troubleshooting techniques are not taught to beginners. Instead, they struggle to find and correct their mistakes because they cannot understand the computer’s feedback.

[39] A term based off the echo command in Linux, which just prints to the console. This technique is basically just using a ton of print statements to tell what is going on and where, and is easily implemented by beginning students. They need practice with printf, anyway.

[40] A teacher might suggest adding print statements or explain that the debugger exists, but lecture time is rarely taken to explain these techniques because they are considered complex for beginning programmers.

[41] Sessions are 50 minutes in length, and a session is rarely spent entirely on using the debugger or other program testing techniques. "Policies & Procedures." The University of Alabama in Huntsville. November 27, 2017. Accessed April 22, 2019. https://www.uah.edu/ssc/tutoring/policies-procedures.

Proposed Solutions based on Foreign Language Pedagogy

Explain Programming in an Active and Engaging Manner

To keep students engaged in programming languages, a blend of deductive and inductive teaching methods should be used to balance instruction and independent learning, just as a blend of these methods is used to teach foreign languages. Deductive methods are those in which “the rules, patterns, or generalizations are presented to the student, and then he or she is given ample opportunity to practice the new feature of grammar”; this method can help students learn a language with better accuracy[42]. Because students are given the correct pattern directly in deductive instruction, this method is well suited to “the presentation of irregular patterns or exceptions to general patterns, for these by their very nature cannot be discovered through analogy” (85). Deductive teaching alone, however, does not encourage student learning in the same way as inductive methods, in which “the teacher first gives the students examples of the grammatical structure to be learned [and] after the examples have been practiced, the students are guided in forming a generalization about the grammatical principle they have been working with” (90). The ultimate goal of the inductive method is that, “after much repetition of pattern drills in phrases or sentences, the grammatical principle [is] to be induced or inferred by the students, or it [will be] given to them by the teacher”[43].

Foreign language learning is most successful when inductive and deductive teaching methods are combined throughout the course. Inductive methods mimic the way in which a baby

[42] Allen and Valette, Classroom Techniques: Foreign Languages and English as Second Language, 1977, 85.

[43] Childers, J. Wesley. *Foreign Language Teaching*. New York: Center for Applied Research in Education, 1965, 31.

learns its first language, without much instruction, and are thus considered a natural way to learn a language (32). Once the student understands grammar (both its purpose and application) from learning their native language, however, their study of a new language can be expedited by presenting grammatical rules deductively. Purely inductive instruction can be frustrating for students trying to learn irregular patterns. To keep students involved and engaged in their own learning process, inductive methods are used; to keep students from being frustrated by difficult patterns and to encourage accuracy in language learning, deductive teaching methods are used. The strength of these two methods lies in their being implemented together.

Programming is traditionally taught very deductively, mostly because this manner of teaching is deemed necessary. Programming is full of complex rules and structures that are difficult for students to learn without instruction. Deductive methods, because they focus more on grammar and correctness, are much more heavily used to teach programming to beginners. Beginning programs still have to be syntactically correct in order to run, and “the curricula content in [programming language classes is] focused on teaching theory of formalized languages and their grammars, theory of automates and their operation systems, and mainly on acquisition knowledge and skills in programming in a particular programming language”[44]. The traditional model of a programming course follows what can be described as the traditional “objectivist” or deductive approach, where “the instructor serves as a subject matter expert who

[44] Zahorec, Jan, Alena Haskova, and Michal Munk. "Assessment of Selected Aspects of Teaching Programming in SK and CZ." *Informatics in Education*13, no. 1 (2014), 159.

is the primary source of knowledge [and] learning is seen as an information transfer process…typified by the lecture”[45].

A few attempts have been made to teach programming inductively, by placing students in a programming environment where they can test code independently, but these were largely unsuccessful. The principle was the same as inductive teaching in foreign language; to simulate the original learning environment of young children, where “they can interact and participate in their own self-directed manner”[46]. However, students learning programming “in this type of unstructured educational environment… do not seem to acquire the "powerful ideas" that are often claimed of [programming] experiences” (78). Induction alone, although typically to how professional programmers learn new languages independently[47], is not successful in teaching programming for beginners, but pure deductive teaching is also not very engaging for students.

Because programming is a language, instruction methods should follow the example of foreign language teaching and employ inductive and deductive teaching methods together. Instead of the typical lecture-based model, some computer science professors like Wulf advocates for a combination of inductive, student-led learning and deductive, lecture-based

---

[45] Wulf, Tom. "Constructivist Approaches for Teaching Computer Programming." *Proceedings of the 6th Conference on Information Technology Education - SIGITE 05*, 2005. doi:10.1145/1095714.1095771, 245.

[46] Palumbo, "Programming Language/Problem-Solving Research: A Review of Relevant Issues." 1990, 78.

[47] They debug it on their own until they can figure it out. Because they already know how to program, they can do this. New programmers have no idea how to, so induction doesn’t work for them.

instruction in a pedagogy he refers to as a “constructivist” approach[48]. Constructivist pedagogy is more student-centric, “taking into account the different characteristics of individual students [to] successfully engage a wider range of learners” (245). Engaging more students with an active blend of inductive and deductive learning is very appealing, especially “in light of the decline in enrollments in computer science programs” (246). Computer science professors, like Wulf, are recognizing that the traditional model is ineffective and unattractive to beginning students and are starting to consider a blend of induction and deduction. This improvement, however, needs to take effect in all introductory programming classes, and not be just an example of one course. Like foreign language, a blend of methods should be part of pedagogy, and not simply a effective option for teachers to consider implementing.

Show How Programming is Useful

Making foreign language learning directly applicable to students’ lives is necessary to foreign language teaching overall. The student must learn grammar rules and vocabulary in order to learn the meaning of the new language, as well as how to express their own ideas. Instead of just words and rules to memorize, then, the student sees vocabulary and grammar as skills necessary to expressing themselves correctly. This difference from simple memorization of rules to an understanding of meaning affects how students complete exercises and assignments, because “there is a crucial difference between practice involving the linking of expression to actual meaning - and practice in which the student’s attention is focused on achieving correctness

[48] Wulf, "Constructivist Approaches for Teaching Computer Programming", 2005, 245.

of expression"[49]. Language practice is more effective when students are focusing more on communication and correct expression, and less on following all the rules.

Pragmatic mapping, the linking of a word and its meaning, occurs first when humans learn their native tongue; to learn a second language, this same pragmatic mapping process must happen again, only now the learner of the language has some concept of what a language is[50]. Active, engaging methods provide students with "concrete referents" - objects or actions that exist in the present[51]. These concrete referents aid students in the pragmatic mapping process by linking language to an experience (43). The words and grammar they are learning are used to directly describe the world around them, and not used merely to construct arbitrary exemplar sentences. Students practicing with concrete referents and meaningful sentences can more readily see how language applies to their life.

Programming language courses often offer programming assignments that sharpen students' coding skills but have no other apparent purpose[52]. Teaching the applications of programming through applicable exercises, however, is crucial to teaching programming languages. Students who do not understand why they are learning programming in the first place will not be motivated to learn for themselves, even if the teacher is using active teaching

---

[49] Oller and Richard-Amato, Methods That Work: A Smorgasbord of Ideas for Language Teachers, 1984, 27.

[50] Ambron, Sueann Robinson., and Neil J. Salkind. *Child Development*. New York: Holt, Rinehart and Winston, 1987, 195.

[51] Oller and Richard-Amato, Methods That Work: A Smorgasbord of Ideas for Language Teachers, 1984, 43.

[52] Hancock, "Context and Creation in the Learning of Computer Programming", 1988, 24

methods. Even if students are taught syntax and semantics together, “many students use code constructs like [for loops] without fully understanding them [and] instead …use them as formulas”(20). Instead of understanding conceptually what a for loop can do, the student will only understand applications of the for loop as it is shown to them and cannot translate this concept into a structure they can adapt and use in varying contexts (20). Teaching programming without teaching the types of problems students can solve with what they are learning leads to students with sufficient knowledge of structures, but no idea how to compose a program to solve a problem using those constructs. Simple programs that sharpen programming skills can of course be useful, but “pure exercise justifies writing one program, it doesn't justify the whole enterprise of programming. Students must also write more meaningful programs”(24) in order to see how programming can be interesting and applicable to their lives.

Historically, computer programming has been taught according to standards defined by the “ACM Computing Curriculum (2001) [which is] based on the mathematical methodology of teaching the programming disciplines… suggesting that programming should be considered as a branch of mathematics”[53]. When presenting computer programming as a mathematical discipline, students may not make the connection that they need to compose their own solutions to problems and will they will then miss out on applying what they are learning outside the classroom. Instead, students will use programming language constructs as a math-like formula, and not like a language construct through which they communicate with a computer. Teaching students in a math-based manner leads to them “[swallowing] the constructs whole, learning

[53] Vitkute-Adzgauskiene and Vidziunas, “Problems in Choosing Tools and Methods for Teaching Programming”, 272.

them as templates instead of as plans”[54]. Template or formula-based learning can limit student creativity and creates a great difficulty for students when they are asked to compose their own solution to a problem (22). Students’ problem-solving skills are limited to formula application skills learned from math class because of the mathematical presentation of coding. While coding does have mathematical elements, it also has a very creative compositional component that is often not emphasized as much as the logical component. In order to teach students to be creative in their problem-solving, programming must be presented more like a language skill, and not just purely mathematical.

Give Students Agency in Selecting a Programming Language

It is important to note that students of foreign language have agency in language selection, and usually learn a foreign language based on some personal motivation. While foreign language as a topic can be a required class, the language itself is often not required specifically, but rather the student may select from a pool of foreign languages taught at their college or university. Whatever their reason for selecting a language may be, the student’s agency can aid in their personal motivation to succeed in the course. Studies have demonstrated that “activating intrinsic motivation produces better performance than extrinsic motivation”; in other words, students interested in learning perform much better than those who simply hope for a good grade[55].

[54] Hancock, "Context and Creation in the Learning of Computer Programming", 1988, 22.

[55] Lin, Yi-Guang, Wilbert J. McKeachie, and Yung Che Kim. "College Student Intrinsic And/or Extrinsic Motivation and Learning." *Learning and Individual Differences*13, no. 3 (December 27, 2002): 251-58. https://doi.org/10.1016/S1041-6080(02)00092-4, 256.

Programming students rarely have the same selection process as foreign language students. Even if multiple programming language courses are offered to beginners, there is no effective way for them to decide which one they are most interested in. Many students, then, simply pick the course that works well with their schedule, or that their friend is taking, or perhaps whichever course is recommended by their advisor. Programming languages do not have some element of culture for students to immediately take interest in, making it difficult to find a way to relate to the student's own areas of interest. Programming languages do, however, have typical applications. C is very useful to students who wish to have more direct control of memory, or perhaps hardware. Python is excellent at manipulation of text data and has extensive libraries that extend its capabilities to innumerable fields.[56] Java is very popular for developing user interfaces and mobile applications. Perhaps, if beginner programming students knew something about the professional applications of each language, they might select a language in which they have some personal interest, and consequently be more motivated to practice independently and to learn about the programming language.

As a concrete example, Python is a scripting language that is more readily used by Earth Systems Science majors at UAH. Several Earth Systems Science students seeking tutoring help for introductory C programming have expressed interest in learning Python. Why, then, if they have a personal and professional interest in Python, were they not encouraged to take Python as their required introductory programming class? If the goal of introductory programming is to gain some knowledge of programming constructs and not just to be familiar with a specific

[56] Python is one of those languages that programmers can basically do anything with, from writing a script to host a webpage to carving out data from a binary file to interfacing with a breadboard. There is a python library for everything.

language, why are students only taking introductory C programming to satisfy catalog requirements? Students should not only be able to satisfy their programming course requirement with one of several different programming language options, but they should also be encouraged to do so.

Provide Rapid Feedback

Foreign language students are encouraged to practice in class extensively. Successful foreign language teaching methods “maximize the number of times each student speaks aloud in the new language” and keep each individual student in the class engaged and enjoying the learning process without excessive homework[57]. One teacher of foreign language, John Rassis emphasizes this volume of practice in the classroom, and tells his students, “‘have the courage to be bad, to make mistakes - but speak!’” (97) By getting students to speak more in class, not only are the students more involved in the class and practicing their target language, but they are also able to receive rapid feedback from the instructor. Correction is not so severe that the student is scared to speak, but rather just enough that the student learns correctly and quickly. Feedback is also provided to the students immediately after they utter a sentence or two. Any problems with vocabulary or grammar can be addressed quickly, before mistakes take root into a student’s understanding of the language.

For programming students, simple debugging techniques allow them to understand what the computer is ‘saying’ in error messages, or to understand what their program is *actually* instructing the computer to do. Communication with the computer allows students to verify if they are using the language correctly; this direct communication with the computer is closer to

[57] Oller and Richard-Amato, Methods That Work: A Smorgasbord of Ideas for Language Teachers, 1984, 95.

the rapid feedback given to a foreign language student by a native speaker. If, after all, a programming language is used to communicate with a computer, the computer is the 'native speaker' of the programming language and is therefore the party the student needs to practice the language with. The only issue with students practicing with this 'native speaker' is that the student must learn how to understand the computer's feedback[58]. Once the student has been given enough context to understand the feedback the computer is providing, the middle-man translator role of the teacher can be removed.

Professional programmers can verify for themselves that their program works using software testing techniques. These techniques, however, are not taught to beginning students because they are considered more advanced topics. If simplified versions of software testing techniques could be taught to beginning students, or if they were given code that would test their solution[59], students would be able to verify for themselves the correctness of their own code. Debuggers are not complex to explain to beginners; they are simply omitted from introductory curriculum because they are somehow deemed too complex or not important enough to mention until much later. This is a disastrous oversight in programming instruction that needs to be immediately rectified.

---

[58] Alternatively, the computer could be instructed how to give more understandable feedback to beginners. A system developed to perform this role was developed and researched by Cardoso, Castro, and Rocha, "Integration of Virtual Programming Lab in a Process of Teaching Programming EduScrum Based", 2018, and found to be effective: the "experience already achieved is promising and that the preliminary results are encouraging".

[59] Such as a test driver, or a function that will run student's code and verify the results. Such a driver could be created by the instructor.

Concluding Remarks

This paper has discussed the similarities between natural language and programming language in history, structure, and usage. The research presented here also pointed out issues programming pedagogy, specifically in the areas of teaching syntax, application, and context, and encouraging meaningful practice. Solutions for these issues were found in foreign language pedagogy and applied to programming pedagogy. Like foreign language courses, introductory programming can explain concepts in an active and engaging manner, explain why programming is useful and applicable to students, provide some agency in language selection, and give students rapid feedback about their progress in language learning through debugging knowledge.

The point of this research is not to disparage introductory programming teachers. Programming is a difficult subject for students to learn, and teachers do not have the same body of programming-specific pedagogical literature as other disciplines like foreign language. This paper is calling attention to a body of literature that is available for programming teachers that could be useful in developing programming pedagogy further; by considering foreign language pedagogy specifically, teachers can find more specific help for teaching programming as a language.

The main reason for suggesting these changes is to encourage student interest in programming. Computer Science is a constructive discipline with interesting and complex problems that require creative solutions. Creative students can devise new and innovative solutions to problems. These creative students will not be drawn to programming, however, if introductory classes are not showing students how programming can be fun and creative. As a computer science student myself, I was drawn to the creative problem-solving aspects of programming, and was surprised when student seeking help at the Student Success Center did not see programming in this light. By completing this research, I hope to bring a new light to the topic of programming pedagogy, in hopes that future students will be as interested in the linguistic component of computer programming and a new application of their compositional skills as I am.

Works Cited

Adams, Heather. "How Students Choose Their Foreign Language Classes." KBIA. April 29, 2014. Accessed April 22, 2019. https://www.kbia.org/post/how-students-choose-their-foreign-language-classes#stream/0.

Allen, Edward David., and Rebecca M. Valette. *Classroom Techniques: Foreign Languages and English as Second Language*. Harcourt Brace Jovanovich, 1977.

Aureliano, Viviane Cristina Oliveira. "A Methodology for Teaching Programming for Beginners." *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research - ICER 13*, 2013. doi:10.1145/2493394.2493417.

Beechhold, Henry F., and John L. Behling. *The Science of Language and the Art of Teaching*. New York: Scribner, 1972.

Cardoso, Marilio, Antonio Vieira De Castro, and Alvaro Rocha. "Integration of Virtual Programming Lab in a Process of Teaching Programming EduScrum Based." *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, 2018. doi:10.23919/cisti.2018.8399261.

Childers, J. Wesley. *Foreign Language Teaching*. New York: Center for Applied Research in Education, 1965.

Halliday, Michael Alexander Kirkwood, and Jonathan James Webster. *On Language and Linguistics*. London: Continuum, 2004.

Hancock, Chris. "Context and Creation in the Learning of Computer Programming." *For the Learning of Mathematics*8, no. 2 (February 1988): 18-24. Accessed January 10, 2019. https://www.jstor.org/stable/40248137.

Lin, Yi-Guang, Wilbert J. McKeachie, and Yung Che Kim. "College Student Intrinsic And/or Extrinsic Motivation and Learning." Learning and Individual Differences13, no. 3 (December 27, 2002): 251-58. https://doi.org/10.1016/S1041-6080(02)00092-4.

Oller, John W., Jr., and Patricia A. Richard-Amato. *Methods That Work: A Smorgasbord of Ideas for Language Teachers*. Taipei: Wen He, 1984.

Rivers, Wilga M. *The Psychologist and the Foreign-Language Teacher*. Chicago: Univ. of Chicago Pr., 1976.

Pal, Kaushik. "Why Is Learning Assembly Language Still Important?" Techopedia.com. Accessed April 22, 2019. https://www.techopedia.com/why-is-learning-assembly-language-still-important/7/32268.

Pears, Arnold, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. "A Survey of Literature on the Teaching of Introductory Programming." *ACM SIGCSE Bulletin*39, no. 4 (2007): 204-23. doi:10.1145/1345375.1345441.

"Policies & Procedures." The University of Alabama in Huntsville. November 27, 2017. Accessed April 22, 2019. https://www.uah.edu/ssc/tutoring/policies-procedures.

Sebesta, Robert W. *Concepts of Programming Languages*. Ed Wood City, Cal.: Benjamin/Gumming, 1993.

Scott, Michael L. *Programming Language Pragmatics*. San Francisco, CA: Morgan Kaufmann, 2004.

Sleeman, Derek. "The Challenges of Teaching Computer Programming." *Communications of the ACM*29, no. 9 (1986): 840-41. doi:10.1145/6592.214913.

Suo, Xiaoyuan. "Toward More Effective Strategies in Teaching Programming for Novice Students." *Proceedings of IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE) 2012*, 2012. doi:10.1109/tale.2012.6360379.

Van Merrienboer, Jeroen J. G., and Hein P. M. Krammer. "Instructional Strategies and Tactics for the Design of Introductory Computer Programming Courses in High School." *Instructional Science*16 (1987): 251-85. doi:10.1007/BF00120253.

Vitkute-Adzgauskiene, Daiva, and Antanas Vidziunas. Problems in Choosing Tools and Methods for Teaching Programming. Vol. 11. Kaunas, Lithuania: Informatics in Education, 2012. PDF.

Zahorec, Jan, Alena Haskova, and Michal Munk. "Assessment of Selected Aspects of Teaching Programming in SK and CZ." *Informatics in Education*13, no. 1 (2014): 157-78.