International Journal of Humanities and Social Science Research
ISSN: 2455-2070; Impact Factor: RJIF 5.22
www.socialresearchjournals.com
Volume 2; Issue 12; December 2016; Page No. 59-62

Syntactic versus conceptual lesson plans: Towards improving middle school computer science curricula

[1] Shaan Bhandarkar, [2] John Leddo, [3] Siddharth Lakkoju, [4] Surya Somayyajula

[1] Phillips Exeter Academy, New Hampshire, USA
[2-4] My Edmaster, Virginia, USA

Abstract
The advent of computers in our society has created the significant profession of computer programming, which is offered as a major in many colleges and universities. Accordingly, there has been much research on what is the most effective way to teach computer programming to college students and other adults. Two common methods include conceptual, where people are taught according to the principles behind programming concepts, and syntactic, where people are taught based on the rules or grammar of the programming language. Much research suggest that syntactic teaching leads to better learning in adults than does conceptual teaching. Nowadays, we find that even children are learning computer programming. Therefore, it is of interest to investigate whether a conceptual or syntactic teaching approach works better with younger learners. In the present project, 37 middle scholars were taught basic Java programming through either conceptual or syntactic instruction. The topics covered were for and while loops. Results showed that students taught using the syntactic method scored higher on a programming post-test than those taught using the conceptual methods. Further research would shed light on whether these results hold for other programming concepts or whether a hybrid approach may produce even stronger learning.

Keywords: syntactical learning, conceptual learning, java programming

1. Introduction

The benefits of conceptual (semantic) learning and syntactic learning in programming have been investigated for many years (cf., Shneiderman and Mayer, 1979) [1]. Early studies report that semantic learning can help grow the programmer's knowledge such as the study done by Shneiderman and Mayer, in which the researchers took novices and added an independent variable (a standard manual) that included conceptual models, so the novices could learn the programming language BASIC. Syntactic learning was also used in complement to grow the programmer's knowledge. Their studies show that syntactic learning can help college students become better programmers. They used a cognitive framework to describe behaviors involved in programming, and they used an information processing model that included syntactic and semantic knowledge. They found experimental evidence to support the model of syntactic/semantic interaction.

Studies report that novices who learn programming using a language (for example, Kodu) can gain significant knowledge of semantics and syntax (mostly semantics). For example, Alice, a programming language that emphasizes concepts and syntax, is a 3-D Interactive Graphics Programming Environment for Windows. The Alice programming language helps programming novices to develop 3-D environments and to explore the medium of 3D graphics, as well as learn programming concepts. In their experiment, Cooper, Dann, and Pausch (2003) [2] taught programming novices with Alice. The experimenters found that the novices were actually quite comfortable using Alice. They were able to find mistakes in their code and debug the code. They also developed their knowledge of programming concepts.

There are programming classes that teach introductory level programming, intermediate level programming, and high level programming these classes focus on syntactic, semantic, and strategic programming knowledge. Some classes are structured in a way so that the class focuses on one aspect of programming knowledge as opposed to combining all three of them. Studies report that programmers need all three aspects of programming knowledge as opposed to just one. For example, McGill and Volet (1997) [3] used a conceptual framework that integrated syntactic, semantic, and strategic programming knowledge with declarative, procedural, conditional knowledge. Analysis of experimental data from a previous experimental study (Volet, 1991) [4] showed the validity of their model. The model can diagnose deficiencies of the programming knowledge of novice programmers through course instruction.

Anthony Robins, Janet Rountree, and Nathan Rountree (2003) [5] from the University of Otago in New Zealand reviewed the literature relating to the psychological/educational aspects of programming. By testing novice and expert programmers with different programming strategies, program generations and comprehensions, the researchers were able to identify different trends in the different programmers. Their research concluded that the novice programmers (who were more syntactical) did not do as well as expert programmers (who were more conceptual). The expert programmers were able to apply their strategies to different tasks they had not seen of before, whereas novice programmers had almost no way of approaching these questions and resulted in failing to complete the tests.

Most of these experiments conclude that novice programmers are more syntactic, while expert programmers are more

conceptual. In these experiments, the students learned through both syntactic and conceptual approaches programming. In general, the students gained a sizable amount of syntactic knowledge of the programming, but not much conceptual knowledge of general programming concepts.

The above research was conducted with adult populations. Nowadays, people are learning to program at a younger age and this has drawn the attention of the research community. For example, Adams and Webster (2011) [6] went to an "after school center" where the kids learned Scratch. The experiment lasted over an 18-month period, in which the students (all of ages 8-18), created 536 projects overall. The experimenters analyzed and drew conclusions from the quality of the projects the kids made. After the 18 month period, the kids developed a better understanding of basic programming concepts (such as variables and conditional statements).

Since students are learning how to program at a younger age, the present study focuses on which method would be more effective for teaching younger people: conceptual or syntactic. Our experiment is novel in that we focus on the effects of semantic vs. syntactic learning on middle scholars as opposed to other groups utilizing other age groups as test subjects. We also focus on a specific programming language (Java**)** as opposed to the many other computer languages in experiments conducted by others. In the present experimental design, two groups of middle school students with no prior programming experience learned basic Java. In this respect, the students could be considered novices as in the above-cited studies. One group was taught using a conceptual approach and the other was taught using a syntactic approach.

2. Materials and Methods

Participants

Participants were 37 middle school students recruited from the Herndon, Virginia area. Participants responded to a flyer advertising a free Java workshop and were not paid for their participation. The flyer was placed at the MyEdMaster tutoring center and also at the local library. Participants were given a pre-test to insure that they had no significant knowledge of the Java programming language.

Materials

To accurately represent both of the approaches to teaching programming languages, we developed two different instructional sets (one conceptual-based and one syntactical-based) followed by the same practice problems and final examination. The conceptual handout was marked by its inclusion of relatable, real world examples of while loops and for loops in our lives as well as emphasized the conceptual understanding of the uses of each loop structure. The syntactical instructional set essentially focused on syntax and numerous instances of actual code implementing the practice problems with no detailed explanations or real life examples. The conceptual and syntactic instructional sets were modeled after the format used by Shneiderman and Mayer (1988) in their studies comparing conceptual versus syntactic instruction.

Procedure

The 37 participants were randomly assigned to an instructional group (syntactic or conceptual) where nineteen were in the syntactical group and eighteen were in the conceptual group. Students were taught in groups of five to twelve students to simulate a standard classroom environment. An instructor taught the programming, following the curriculum in the instructional set. Instruction lasted for about 30 minutes, after which students were given a set of four practice problems to work on. The practice problems were the same for both groups and included two while loop coding problems and two for loop coding problems. The practice problems were untimed and took about 15 minutes to complete. To prevent the use of any online academic resources by the students and consequent skewing of the results, all the coding was done on paper. Upon completion of the four practice problems, students were given individual feedback on their solutions. Students were allowed to ask questions throughout the entire instructional process.

After the instruction and practice problems, students were given a timed 45-minute, fifteen point, six question test comprised of four free-response questions each worth two points, one short-answer question on the output of loops worth four points, and one short-answer question on the output of Boolean operations worth three points. The grading rubric for the free-response questions was constructed based roughly on the penalty for errors as defined in the 2010 AP Computer Science General Free Response Scoring Guidelines ("AP® Computer Science A 2010 Scoring Guidelines", 2010) [7] and the 2015 AP Computer Science Question Specific Free Response Scoring Guidelines ("AP Computer 2015 Computer Science A Guidelines", 2015) [8] and is shown in Table 1.

Table 1: Grading Rubric for Java Final Exam

Penalty	Error
-0.25 points	Missing semicolon if majority of them are there, misspellings, redundant data type declaration, redundant semicolon, invalid variable name
-0.5 points	Incorrect variable declaration, invalid syntax for termination condition in a while loop
-0.75 points	Undeclared variable, right pieces of loop structure in incorrect order,
-1 points	Correct loop structure, but incorrect logic
-1.5 points	Correct logic with minor imperfections in loop structure
-2 points	Incorrect logic and incorrect loop structure
Penalty	Error

In order to avert any potential grading bias, we asked that the students not write their names on their papers.

3. Results and Discussion

Students' responses on the final test were scored in

accordance with the rubric presented above. The highest possible score on the post-test was 15. The mean post-test scores for each condition are shown in Table 2.

Table 2: Mean Scores for the Java Final Exam

Syntactic Group	Conceptual Group
11.5	9.2

Table 2 shows a difference in mean post-test scores between the two groups with students in the syntactic group scoring higher than those in the conceptual group. To determine whether this difference is statistically significant, a two-tailed t-test was performed. The results show that the two means are different, $t = 4.36$, $df = 35$, $p < .001$. This suggests that middle school students learned the Java concepts of for and while loops better using syntactic instruction than they did using conceptual instruction. These results mirror earlier findings that adult learners learned computer programming better from syntactic-based instruction than they did from semantic-based instruction.

4. Conclusion and Recommendations

We can conclude that the syntactical lesson did indeed lead to superior performance on the final Java examination compared to the conceptual one. Teachers may still want to teach the logic behind the syntax of any programming language, but it is imperative to develop a form of "pseudocode" and teach the actual syntax with lots of short practice coding exercises concurrently. Testing the efficacy of adding a conceptual component to the syntactic instruction is an area worthy of further research.

We note that our experimental results are limited to people with no prior programming experience who were learning the basics of Java for the first time. In essence, these students were novice programmers. Accordingly, it makes sense that novice programmers would respond to syntactic instruction more than they would to conceptual instruction. According to John Anderson's ACT* theory (1983) [9], learning involves transforming facts into procedures or rules for execution. A syntactic teaching approach is a means to provide those rules. On the other hand, research by John Leddo and his colleagues (Leddo *et al*., 1990) [10] suggests that becoming an expert in a field involves translating procedures into more generalized problem solving skills. This suggests that a conceptual instructional approach would be a useful follow on to syntactic instruction once a student has demonstrated mastery of the syntax and basic plans/schemas for programming. Testing this hypothesis would be a useful continuation of the present research.

The applicability of our thorough approach, trying out different methods of and statistically isolating the most effective one, goes far beyond students just learning Java. In fact, we could apply this to students wishing to learn C/C++, Python, other object-oriented programming languages, and app development languages too. We could even optimize the usefulness of different lesson plans for foreign languages as well, since these have both conceptual and syntactic components.

The extension to other languages, both computer and natural, opens up a new line of research. Computer programming as a profession has the interesting property that programmers tend to be "polyglots" in that they learn multiple programming languages. Similarly, people who learn foreign languages typically have as a starting point their native language. Languages are often conceptually similar but syntactically different. The concept of a for loop exists in virtually every programming language, but the syntax is typically different. Therefore, we might expect learning a for loop in a new language to be facilitated more by having a conceptual understanding of for loops in general than a syntactic understanding of for loops in a different language that likely also has a different syntax. This creates the research question of how to teach computer programming in a way that not only facilitates learning the initial language, but also facilitates transfer of that knowledge to new programming languages.

In any case, an amalgam of physically trying out different lesson plans and numerically analyzing the efficacy of each plan can open new doors in creating better lessons plans for the future of education.

5. References

1. Shneiderman B, Mayer R. Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results [Scholarly project]. In Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results. Retrieved August 11, 2016, from https://cs.umd.edu/~ben/papers/Shneiderman1979Syntactica.pdf. 1977.
2. Cooper S, Dann W, Pausch R. Teaching objects-first in introductory computer science [Scholarly project]. In ACM Digital Library. Retrieved August 11, 2016, from http://dl.acm.org/citation.cfm?id=611966. 2003.
3. McGill T, Volet S. A Conceptual Framework for Analyzing Students' Knowledge of Programming [Scholarly project]. In Research Gate. Retrieved August 11, 2016, from https://www.researchgate.net/publication/43980369_A_Conceptual_Framework_for_Analyzing_Students'_Knowledge_of_Programming. 1997.
4. Volet SE. Modelling and coaching of relevant metacognitive strategies for enhancing university students' learning. Learning and Instruction. 1991; 1:319-336.
5. Robins A, Rountree J, Rountree N. Learning and Teaching Programming: A review and discussion [Scholarly project]. In Learning and Teaching Programming. Retrieved August 11, 2016, from http://home.cc.gatech.edu/csed/uploads/2/robins03.pdf. 2003.
6. Adams JC, Webster AR. *What Do Students Learn About Programming From Game, Music Video, And Storytelling Projects?* [Scholarly project]. In *What Do Students Learn About Programming From Game, Music Video, And Storytelling Projects?* Retrieved August 11, 2016, from http://delivery.acm.org/10.1145/2160000/2157319/p643-adams.pdf?ip=71.163.227.128&id=2157319&acc=AUTHOR-IZED&key=4D4702B0C3E38B35.4D4702B0C3E38B35.4D4702B0C3E38B35.E1E6E872C3B249EA&CFID=824401055&CFTOKEN=33204660&__acm__=1470936356_c3bfbaed1fc10d8c2575f85d2d071931. 2011.

7. AP® Computer Science A. Scoring Guidelines. (2010). Retrieved August 30, 2016, from http://apcentral.collegeboard.com/apc/public/repository/ap10_comp_sci_a_scoring_guidelines.pdf 2010.
8. AP® Computer Computer Science A Guidelines. (2015). Retrieved August 30, 2016, 2. from https://secure-media.collegeboard.org/digitalServices/pdf/ap/ap15_comp_sci_sg.pdf. 2015.
9. Anderson JR. The architecture of cognition. Cambridge, MA: Harvard University Press. 1983.
10. Leddo J, Cohen MS, O'Connor MF, Bresnick TA, Marvin FF. Integrated knowledge elicitation and representation framework (Technical Report 90-3). Reston, VA: Decision Science Consortium, Inc. 1990.