Reducing the Dropout Rate in an Introductory Programming Course

• Aharon Yadin •

This article describes an action research for reducing the high students' dropout rate after an introductory programming course. As part of the action research, that was performed during four semesters several course structures and learning tactics were examined. The success was attributed to three main factors. (1) using Python as the first introductory programming language, which freed the students from detailed language syntax and allowed them to concentrate on algorithms and problem solving; (2) using a visualization environment (Micro-world) for the whole duration of the course, which helped in understanding the more complex and abstract issues; and (3) using individual assignments that enforced better learning habits. The article describes the various attempts, as well as the final structure that reduced the failing students by over 77%.

1 INTRODUCTION

This article describes an action research for establishing the right structure of an introductory programming course that will reduce the high dropout rate.

With the rapid advancement of technology, in recent years, education has been transformed from mainly a teaching discipline to a technology enabled learning environment. Education has shifted from delivering content to the students to a continuous process in which the students acquire facts and theories and build the conceptual models representing their understanding (Dillon, 1987). These models represent one of the cornerstones for the problem-solving skills required in modern society and which are an important part of the introductory courses' outcome. This paradigm shift, from teaching to learning, which started over a decade ago, was addressed by many scholars (Barr and Tagg, 1995; Bell and Lane, 1998; Lenschow, 1998; DuFour *et al.*, 2005) and was influenced by the understanding that for effective learning, the students have to construct their own knowledge. As such, at present, learning is mainly viewed as a process in which students explore and enhance their experience and knowledge. These new perceived experiences, combined with existing knowledge construct new layers of understanding, thus changing and renewing the existing learners' conceptual models. The traditional learning environment has changed as well and by using technology is not confined only to the classroom.

The continuous process of learning, including adapting and enhancing the conceptual models, occurs in a variety of learning locations and even in virtual environments representing the real world.

The constructivist model is a learner-centered process in which the learning responsibility relies on the students. In achieving the learning goals, students are involved in both group and individual learning activities.

2 LEARNING THEORIES

There are many learning theories that have been developed over the years; however, the one that is widely used is the constructivism theory, which is based on Piaget's theory of children's development. According to Piaget, information and data are perceived and maintained using "mental structures" that represent knowledge. Learning is the process of comparing the existing mental structures with the new received information in order to assess its validity. If the new received information makes sense, it will be integrated into the existing mental structure (or accommodated in Piaget's terms). This process of accommodation changes and renews the mental structure and is the cause of learning. If, on the other hand, the new information is very different, does not fit or contradicts the mental structure, it will be rejected or changed so that it will fit the structure. If students are forced to "understand" the new information, but if it does not fit their mental structure, they will memorize it without proper understanding, which implies that it is not conceptualized and will not be used in future problem-solving. Since the constructivism theory defines learning as integration of new experiences with the past mental structures, learning means changing these previous models with relevant new information (Zhi-Feng *et al.*, 2001). Since the early 1970s, cognitive researchers (Anderson, 1980; Squire, 1987; Johnson, 1995; Ten Berge and Van Hezewijk, 1999; Biggs, 2003) distinguish between two types of knowledge: declarative and procedural. Declarative knowledge (also referred to as propositional knowledge) is defined as factual information ("knowing that"), while procedural knowledge ("knowing how") is about how to perform a specific task, or the skills required to operate in the environment (Ten Berge and Van Hezewijk, 1999). In identifying the proper teaching mechanisms, the teacher has to define the learning outcomes and choose the proper activities to promote the acquisition of the two types of knowledge. The shift from teaching to learning is based on the understanding that teaching is not just transmission of information from the teacher to the students (declarative knowledge), but rather, it should be used to create the various relevant activities that will stimulate students and help them construct their own mental models representing meaning. Using this learning theory, the teacher has to define the learning environment, including activities, methods and assignments, so that it will enable the students to acquire the required knowledge.

The constructivist model is a learner-centered process in which the learning responsibility relies on the students. In achieving the learning goals, students are involved in both group and individual learning activities. Many researchers have reported that group learning helps the students build their understanding faster and more efficiently (Beckman, 1990; Cooper *et al.*, 1990; Goodsell, *et al.*, 1992). Group learning has had many different names such as collaborative learning, peer learning, team studying, collective learning, and study or work group. However, according to Johnson *et al.* (1991), we can categorize all of these learning methods by three general types. They include (1) informal learning groups – formed ad hoc for addressing a specific one-time learning session; (2) formal learning groups – formed for a specific task, with a longer duration (like a project) requiring several meetings; and (3) study teams – formal learning groups, working together for an even longer duration (whole semester, or the whole academic year). Study teams usually form a social group in which the relationships extend the study sessions. However, although the study group and its social interaction form a supportive learning environment, the learning (or the changes in the mental structures) and attaining knowledge remains an individual process. For that reason, some researchers suggest structuring courses on collaborative and cooperative study teams with emphasis on individual responsibility and accountability (Prince, 2004). This and the introduction of technology supported collaborative learning systems imply that students have to be more autonomous in their learning attitude (Webster and Sudweeks, 2006).

3 INTRODUCTORY PROGRAMMING COURSES

Undergraduate introductory computer science (ICS) courses are often perceived by the students as problematic based on the relatively high dropout rates. Furthermore, the programming skills acquired by the students after successfully completing these courses are often not sufficient (Nikula et. al., 2007). This is not a new issue and it has been addressed by debates among many researchers, scholars and educators. One of the causes is the high degree of abstraction and complexity required when dealing with the programming paradigm concepts (Robins, Rountree, & Rountree, 2003; Rich, Perry, & Guzdial, 2004). Others suggest that the introductory courses have to concentrate not only on the programming concepts, but also on addressing algorithmic thinking and train students on ways to find solution to problems (Forsyth, et.al. 1975). This approach uses an even higher level of abstraction, ignoring the programming language and focuses on building and enhancing the capabilities of constructing the required algorithm (Miller and Ranum, 2005), or enhancing the mental models, as the constructivism theory defines it. This debate on finding the best ways to tackle successfully the introductory courses is fueled by the fact that although the market recovered from the problems caused by the burst of the dot.com bubble, the enrollment remains low. The decreased interest in the CS (Computer Science) discipline (Nikula, 2007; Radenski, 2006) combined with the very high (up to 50%) dropout rates (Guzdial, 2003; Rich Perry, & Guzdial, 2004; Herrmann *et al.*, 2003; Nagappan *et al.*, 2003) led to various attempts to solve the problem. In dealing with the students' difficulties, several researchers claim that some of the modern programming languages used for introductory courses require the understanding of advanced concepts at an early stage of the learning process. This means that the factual information required by the introductory programming courses interferes with the procedural knowledge. Students who cannot cope with this early understanding are failing the course, not because they do not understand the programming concepts, but because the programming language that was chosen in inappropriate (Miller and Ranum, 2005). As a result, many academic institutes have changed the programming language used as part of the introductory courses. This is not a new trend as was stated by Knuth (2005); he examined the usage of programming languages over the past forty years and concluded that the favorite programming language seems to change every decade. The programming languages used for teaching changed as well and in the past several decades, Scheme, Pascal, C, C++ and Java were the among the popular ones. In recent years, Python is gaining popularity as the first programming language used in introductory courses (Zelle, 1999; Donaldson, 2003; Jenkins, 2004; Radenski, 2006; Goldwasser and Letscher, 2008; Sanders and Langford, 2008 to name a few). Relating to cognitive understanding, the Python programming language is simple, thus its declarative knowledge is minimal, allowing the students to concentrate on the procedural knowledge, which leads to better understanding and enhancing their ability to solve problems. In addition to changing to a more friendly programming language, some researchers and educators started using visual environments in order to help in understanding some of the abstract concepts related to programming and problem solving. The visualization approach for enhancing students' understanding is not new and it has been used to teach children in the late 1970s, for example by using LOGO (Feurzeig & Lukas, 1972; Fischer, 1973; Rubinstein, 1974; Cannara, 1976). These visualization tools and methodologies later were addressed as learning by example or Micro-worlds (Papert, , 1980; Dagdilelis and Satratzemi, 2001; Hoyles, Noss & Adamson.2002; Sarama & Clements, 2002). These Micro-worlds are small, interactive and dynamic learning environments that represent a conceptual model of some part of the real world.

In addition to changing to a more friendly programming language, some researchers and educators started using visual environments in order to help in understanding some of the abstract concepts related to programming and problem solving.

The model usually simplifies the real world and makes it more understandable by providing various tools to explore or manipulate it (Hogle, 1995). The reason for implementing such mechanisms in which first children and later students could develop algorithms without the usage, or knowledge of formal programming language was explained by Eric Roberts: "In real-world programming languages like C, there are so many details that learning about them tends to dominate the first few weeks of a programming course. All too often, they become the focus of the course, and the much more critical issues of problem solving get lost in the shuffle" (Roberts, 1995). The learning by example puts a greater emphasis on the learning, which leads to developing the right algorithms, instead of mastering the specifics of a particular programming language.

The fast technological advancements brought a wealth of additional new tools that were aiming to solve the same problem by defining and designing a friendlier or a gentler approach for teaching programming. "Karel the Robot" (Pattis, 1981) was introduced for teaching Pascal. This is a non threatening, visual environment with a robot living in a two dimensional world (Micro-world) and performing tasks that emphasize programming logic. By instructing the robot to perform the various tasks, the student is gradually exposed to the principles of a programming language and problem solving such as logical deduction and reasoning. The Karel environment was later migrated to support additional programming language, especially Java (Becker, 2001; Buck and Stucki, 2001; Bergin, et.al. 2005) and Python.

4 THE STUDY

The current action research was performed during the last four semesters as part of the ICS course. The course is delivered during the first semester of the first year and represents the primary encounter students have with programming, logic and problems solving. ICS is intended to lay the foundation for the later more complex courses, however for students with no prior programming knowledge it represented a significant challenge. Since 2001, our college has implemented the "objects first" approach and Java was used as the first programming language. However, we decided to change because of the relatively low student success rate. Other contributing factors included the increasing number of papers favoring the procedural (or imperative) first approach (Kölling, 1999; Pillay and Jugoo, 2005; Nesbit, 2009) and the many researchers addressing various problems associated with the objects first approach. The main reason for the latter is because object-oriented programming is hard for students to understand (Bruce *et al.*, 2004; Eckerdal & Berglund, 2005; Mannila, Peltomäki, & Salakoski, 2006; Robins, Rountree, & Rountree, 2003 to name a few). Our current ICS course is concentrating on procedural programming, while the next programming courses concentrate on the object-oriented paradigm. During 2009, the course was taught in both semesters and on 2010 and 2011 only during the first semester. The total number of students enrolled is relatively small and in addition there was a large fluctuation in this number, as demonstrated by Figure 1.

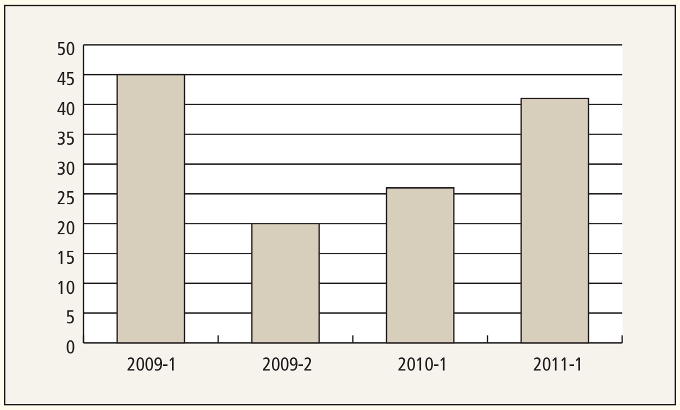

Figure 1: Number of students per semester

The problems associated with the course are similar to problems reported by other academic organizations, i.e. a high dropout rate and the students who completed the course posses lower than expected programming and problem solving capabilities. Originally and for a long time, the course structure was simple. A three-hour lecture using Java as the first programming language and a two hours lab exercise. However, due to experience gained from other academic institutes (Zelle, 1999; Donaldson, 2003; Jenkins, 2004; Radenski, 2006; Goldwasser and Letscher, 2008; Sanders and Langford, 2008) and based on the students' difficulties; the programming language used was changed to Python. The aim was to concentrate more on algorithms and problem solving (procedural knowledge) and less on the language syntax and constructs (declarative knowledge). The understanding that in our rapidly advancing technological environment in which new programming languages emerge coupled with concentrating on important issues related to programming and problem solving concepts rather than detailed syntax of a specific language supported this change.

For lowering the understanding barriers and helping students construct their mental models that represent knowledge, an additional change was introduced. We added a new support course in parallel to the ICS course. The support course was a two-hour lecture and lab, using "Karel the Robot" Micro-world. The intension was to strengthen the algorithmic capabilities and provide a visual and easier way of understanding, applicable mainly for the more abstract issues such as nested loops, nested conditions and recursion. Many academic institutes use Micro-world environments as part of their introductory programming courses. However, unlike other institutes that use the Karel environment just for preliminary understanding, mainly during the first one or two lessons, the structure we used was based on a semester long usage. This way Karel was used not only for programming constructs, but also for visualizing some of the more complex concepts and especially designing and checking various algorithms for solving problems. Although students worked individually, the lab acted as a formal learning group during the whole semester, in which the students worked individually, but learned collectively.

Unfortunately, the new course structure that involved changing to Python and adding a visualization tool, a move that helped students in other academic institutes, had no positive effect in our case and the percentage of the failing students remained high. During the two semesters in 2009, in which this structure was employed, the failing percentages were similar and very high (see Figure 2). A thorough analysis which included discussions with students regarding their difficulties revealed that "Karel the Robot", which initially was considered a visualization tool for enhancing understanding, caused more confusion. The course lectures concentrated on teaching procedural programming, while Karel is using an object oriented approach. This difference not only did not provide the required assistance, but it even caused more misunderstanding. Furthermore, although the two courses (ICS and the support course) were two parts of the same course, they were delivered by two lecturers, which may have caused additional confusion. Another troubling issue linked to Karel, was related to the fact it proved to be an unstable environment, which may suddenly abort, without saving the current project. Due to the course structure, in which Karel was used throughout the whole semester, the stability issues became of a great importance, unfortunately with a negative impact. During the first half of the semester, while the examples and exercises were relatively simple, everything worked fine. However, during the second half, when the exercises are more complicated and the students had to define many new classes the environment became unstable. This problematic behavior translated into many lost hours and turned into a frustrating issue. Consequently, some students preferred to stop using the environment, even at the expense of a lower grade.

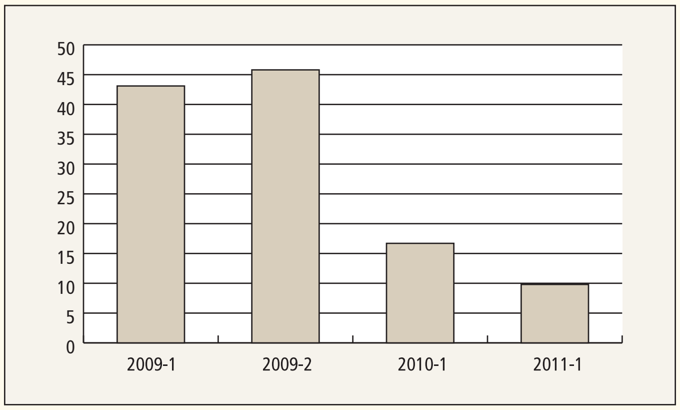

Figure 2: Students' Failing Percentage

Based on Python's success in other institutes, the decision was made to continue using Python as the first programming language, and replace the supporting visualization environment. On the third semester, "Karel the Robot" was replaced by GVR (Guido Van Robot) a Python based implementation of Karel. This open source product supports a variety of platforms where one can download it freely and install it on the students' computers. As part of the preparations for the course a long and intense benchmark was carried and several problems that were discovered in the product were corrected. For enhancing understanding the two courses were delivered by the same lecturer, which allowed for better integration between the two courses and relating smoothly from on course concepts to the other. This change was extremely successful and the number of failing students, in this version of the course, was reduced by 63.5%, from 45.8% of failing students to 16.7% (Figure 2).

The last version of the course was very similar, with only one change. The support course (the GVR Micro-world), which included several assignments and contributed 10% to the ICS course grade was changed to use individual and personal assignments (Yadin and Or-Bach, 2008). This type of assignments is based on individual assignments, which means that the students cannot share or borrow solutions with/from their friends. Each exercise is unique, so students can only discuss among themselves the algorithms; since each student receives a different assignment one student solution is irrelevant to the other. This change was successful and reduced failing students' percentage by additional 41.6%, from 16.7% to 9.8% (Figure 2).

5 RESULTS AND DISCUSSION

This article describes an action research that was performed in order to help students cope better with the difficulties related to introductory programming courses. The original structure, which used Java and was based on a standard three hours lecture followed by a two hours exercise, was slightly modified. The programming language was replaced by Python and for enhancing students understanding, an additional two hours Micro-world lecture/lab was introduced. During the four semesters of this action research, the students' failing percentage was dropped by 77.4% (from 43.1% to 9.8%). The same lecturer taught all four semesters, so personal traits have a minimal impact.

The issues raised by this action research support similar findings presented in other papers related to the success achieved when changing the first programming language. Furthermore, adding a visualization environment (Micro-world) improved the students' operational knowledge. The net result was that the students enhanced their mental models by developing abstract knowledge related to programming concepts and algorithms for solving problems, instead of concentrating on syntax issues. The use of the GVR Micro-world provided additional insight into the process. The importance of visual environments especially when dealing with abstract concepts is not new; many researchers (Papert, 1980; Dagdilelis and Satratzemi, 2001; Hoyles, Noss & Adamson, 2002; Sarama & Clements, 2002 to name a few) already addressed it. However, this action research demonstrated the importance of these environments and a direct link between them and the actual ICS course. The 77.4% improvement in the dropout rate may be attributed to the fact we used the Micro-world environment during the whole the semester, while, in many academic institutes, where Micro-worlds are integrated into the ICS course they are being used only for the first one or two lectures.

The impact of using the Micro-world was intensified by the fact it created a semester long team based collaboration. Although each student had to work individually on his/her assignments, in the lab, there were sub-groups who worked and learned together, as was evident by the fact they used same seats throughout the whole semester. This supports similar findings by many researches that group learning helps students build their understanding more efficiently

(Beckman, 1990; Cooper *et al.*, 1990; Goodsell, *et al.*, 1992). The lab exercises provided an additional way of collaboration, since it acted as a foundation for discussions regarding various solutions and the benefits and shortcomings of each one. The success attributed to using individual, unique assignments support similar findings, and it contributed to further lowering the failing rate.

The reasons behind the fluctuations in the number of enrolled students are unknown. It may, however, be related to the high percentage of failing students. This is a relatively small regional college and the information, especially in the social networks era is spreading fast. There is some correlation between the failing percentage and the reduction in the enrollment. However, this issue will have to be monitored in the future, before it will become conclusive. **Ir**

References

[1] Anderson, J. R. (1980) *Cognitive Psychology and its Implications*, San Francisco: Freeman.

[2] Barr, R. and J. Tagg (1995) "From Teaching to Learning-a New Paradigm for Undergraduate Education", *Change Magazine*, Nov/Dec, pp. 13-25.

[3] Becker. B (2001): Teaching CS1 with Karel the Robot. *Proc. ACM SIGCSE 32nd Technical Symposium on Computer Science Education*, Charlotte NC. 50-54 ACM Press.

[4] Beckman, M. (1990) "Collaborative Learning: Preparation for the Workplace and Democracy", *College Teaching*, 38(4), 128-133.

[5] Bell, S. and A. Lane. (1998) "From Teaching to Learning: Technological potential and sustainable, supported open learning", *Systemic Practice and Action Research*, 11(6), pp. 629-650.

[6] Bergin, J., Stehlik, M., Roberts, J., Pattis, R: Karel J. (2005).Robot: A Gentle Introduction to the Art of Object Oriented Programming. Dream Songs Press. http://csis.pace.edu/~bergin/KarelJava2ed/Karel++JavaEdition.html. Accessed January 2011.

[7] Biggs, J.B. (2003) *Teaching for Quality Learning at University*, Second edition, Buckingham, Open University Press, Open University Press/Society for Research into Higher Education Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. Computer Science Education, 13(2), 137-172.

[8] Bruce, C., Buckingham, L., Hynd, J., McMahon, C., Roggenkamp, M., & Stoodly, I. (2004). Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university. *Journal of Information Technology Education, 3*, 143 - 160.

[9] Buck, D., Stucki, B (2001): JKarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum. *Proc. ACM SIGCSE 32nd Technical Symposium on Computer Science Education*, Charlotte NC. 16-20 ACM Press.

[10] Cannara, A. B. (1976). "Experiments in Teaching Children Computer Programming." Technical Report No. 271. Institute for Mathematical Studies in the Social Sciences, Stanford University, 1976.

[11] Cooper, J. (1990) "Cooperative Learning and College Teaching: Tips from the Trenches", *Teaching Professor*, 4(5), pp.1-2.

[12] Dagdilelis, V., and Satratzemi, M. (2001). Post's Machine: A Didactic Microworld as an Introduction to Formal Programming, *Educational and Information Technologies*, 6(2), 123-141

[13] Dillon, A. (1987) "Knowledge Acquisition and Conceptual Models: A Cognitive Analysis of the Interface" in Diaper, D. and R. Winder (eds.) *People and Computers III*, Cambridge, UK: Cambridge University Press, pp. 371-379.

[14] Donaldson, T. (2003). Python as a first programming language for everyone. Western Canadian Conference on Computing Education, 2003.

[15] DuFour, R., R. Eaker, and R. DuFour (2005) *On common ground: The power of professional learning communities*, Bloomington, IN: Solution Tree.

[16] Eckerdal, A., & Berglund, A. (2005). *What Does It Take to Learn 'Programming Thinking'?* In Proceedings of the 1st International Computing Education Research (ICER) Workshop, Seattle, WA, USA, 135 -143.

[17] Feurzeig, W., & Lukas, G. (1972). Logo: A programming language for teaching mathematics. *Educational Technology*, March, 1972.

[18] Fischer, G. (1973). *Material and ideas to teach an introductory programming course using Logo*. Irvine, Calif.: Department of Information and Computer Science, U. C. Irvine, 1973.

[19] Forsyth, A. I, Keenan, T. A. Organick, E. I., and Stenberg W. (1975), *Computer science: A first course*, John Wiley, 1975.

[20] Goldwasser. M. H., and Letscher, D. (2008). Teaching an Object-Oriented CS1 - with Python. In *ITiCSE'08*, pages 42-46, 2008.

[21] Goodsell, A., M. Maher, V. Tinto, L. Smith, & J. MacGregor (eds.) (1992) *Collaborative Learning: A Sourcebook for Higher Education*. University Park: National Center on Postsecondary Teaching, Learning, and Assessment, Pennsylvania State University.

[22] Guzdial, M. (2003) Media Computation Course for Non-Majors. *ITiCSE Proceedings*. P104-108. ACM. New York.

[23] Herrmann, N., Popyack, J. L., Char, B., Zoski, P., Cera, C. D., Lass, R. N., *et al.* (2003). Redesigning introductory computer programming using multi-level online modules for a mixed audience, *34th SIGCSE technical symposium on computer science education* (pp. 196-200). Reno, Nevada, USA: ACM Press.

[24] Hogle, J. (1995). Computer Microworlds in education: Catching up with Danny Dunn. (ERIC Document Reproduction Service No. ED 425738). http://www.eric.ed.gov/ERICWebPortal/contentdelivery/servlet/ERICServlet?accno=ED425738 Accessed February 2011

[25] Hoyles, C., Noss, R. & Adamson, R. (2002). Rethinking the Microworld idea. *Journal of Educational Computing Research*, 27(1&2), 29-53.

[26] Jenkins, T (2004). *"The First Language – A Case for Python?" In The 4th Annual LTSN-ICS Conference*. Galway, Ireland August 2004

[27] Johnson, D. W., R.T. Johnson, and K.A. Smith (1991) *Cooperative Learning: Increasing College Faculty Instructional Productivity. ASHE-FRIC Higher Education Report No.4*, Washington, D.C.: School of Education and Human Development, George Washington University.

[28] Johnson, K. (1995) *Language Teaching and Skill Learning*, Oxford, Basil Blackwell.

[29] Kölling, M., (1999). "The Problem of Teaching Object-Oriented Programming, Part 1: Languages," *Journal of Object-Oriented Programming*, 11(8): 8-15, 1999.

[30] Knuth, D. E. (2005). *Art of computer programming*, Volume 1, Fascicle 1, The: MMIX -- A RISC computer for the new millennium. Addison Wesley Professional.

[31] Lenschow, R. J. (1998) "From Teaching to Learning: A Paradigm Shift in Engineering Education and Lifelong Learning", *European Journal of Engineering Education*, 23(2), pp. 155-161.

[32] Mannila, L., Peltomäki, M., & Salakoski, T. (2006). What about a simple language? Analyzing the difficulties in learning to program *Computer Science Education*, 16(3), 211 - 227.

[33] Miller, B. N., and Ranum, D. L. (2005)., Teaching an introductory Computer Science Sequence with Python. Proceedings of the 38th Midwest Instructional and Computing Symposium, Eau Claire, Wisconsin, USA, 2005.

[34] Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., *et al.* (2003). Improving the CS1 experience with pair programming, *34th SIGCSE technical symposium on computer science education* (pp. 359-362). Reno, Nevada, USA: ACM Press.

[35] Nesbit, T. (2009) T*he Teaching of Introductory Programming: Issues of Context*. Napier, New Zealand: NACCQ 2009 22nd Annual Conference, 10-13 Jul 2009. In Proceedings of the 22nd National Advisory Committee on Computing Qualifications, 71-78.

[36] Nikula U., Sajaniemi J., Tedre M., Wray S. (2007) Python and Roles of Variables in Introductory Programming: Experiences from three Educational Institutions. Journal of Information Technology Education, 6, 199-214. *Available at http://jite.org/documents/Vol6/JITEv6p199-214Nikula269.pdf* Accessed January 2011

[37] Papert, S., (1980). *Mindstorms: Children, Computers and Powerful Ideas*, Nova York: Basic Books

[38] Pattis, R.E. (1981): *Karel the Robot: A Gentle Introduction to the Art of Programming*. John Wiley & Sons Inc. New York, NY, USA.

[39] Pillay N., and Jugoo V. R.,(2005). An Investigation into Student Characteristics Affecting Novice Programming Performance, in inroads - ACM SIGCSE Bulletin, Vol. 37, No. 4, ACM Press, December 2005.

[40] Prince, M. (2004) "Does Active Learning Work? A Review of the Research", *Journal of Engineering Education*, 93(3), pp. 223-231.

[41] Radenski, A. (2006). "Python first": A lab-based digital introduction to computer science, *11th annual SIGCSE conference on innovation and technology in computer science education* (pp. 197-201). Bologna, Italy: ACM Press.

[42] Rich, L., Perry, H., & Guzdial, M. (2004). A CS1 course designed to address interests of women, *35th SIGCSE Technical Symposium on Computer Science Education* (pp. 190-194). Norfolk, Virginia, USA: ACM Press.

[43] Roberts, E. S., (1995). *The Art and Science of C: A Library-Based Approach*, Reading, MA: Addison-Wesley, 1995

[44] Rubinstein, R (1974). *Computers and a liberal education: Using Logo at the undergraduate level*. Irvine, Calif.: Department of Information and Computer Science, U. C. Irvine, 1974.

[45] Sanders. I. D., and Langford. S. (2008). Students' Perceptions of Python as a First Programming Language at Wits. In *ITiCSE'08*, page 365, 2008.

[46] Sarama, J. & Clements, D. (2002). Design of Microworlds in mathematics and science education. *Journal of Educational Computing Research*, 27(1&2), 1-5.

[47] Squire, L.R. (1987) *Memory and Brain*. New York: Oxford University Press.

[48] Ten Berge, T. and R. Van Hezewijk (1999) "Procedural and Declarative Knowledge: An Evolutionary Perspective", *Theory and Psychology*, 9(5), pp. 605-624.

[49] Webster, R. and F. Sudweeks (2006) "Enabling Effective Collaborative Learning in Networked Virtual Environments", *Current Developments in Technology-Assisted Education*, (2) pp.1437-1441.

[50] Yadin, A and Or-Bach, R., (2008). Fostering individual learning: when and how. ACM SIGCSE Bulletin 40(4): 83-86

[51] Zelle, J. M., "Python as a First Language," Proceedings of the 13th Annual Midwest Computer Conference. Whitewater, Wisconsin, USA. March 1999.

[52] Zhi-Feng, E., S. Liu, C. Chi-Huang, and Y. Shyan-Ming (2001) "Web-Based Peer Review: The Learner as Both Adapter and Reviewer", *IEEE Transactions on Education*, 44(3), p. 246.

AHARON YADIN
Management Information Systems Department
The Max Stern Academic College of Emek Yezreel
Emek Yezreel, 19300, Israel
aharony@yvc.ac.il

Categories and Subject Descriptors: K.3.2 [Computers and Education]: Computer and Information Science Education – *computer science education, information systems education.*
General terms: Human Factors
Keywords: Reducing dropout, introductory programming course

DOI: 10.1145/2038876.2038894