PyTut: A Lightweight Web-Based Python Tutor

Carla Kirk-Cohen
University of Cape Town
Rondebosch
Cape Town
carlakirkcohen@gmail.com

ABSTRACT

E-Learning is an ever expanding field in computer science education. The dynamic characteristics of programming are particularly suited to interactive e-learning methods. The suitability of E-Learning for programming courses is discussed in this paper. It is concluded that programming is particularly suited to e-learning and that the reduction of barriers to learning such as the use of IDEs is important for encouraging and nurturing beginner programmers. The benefits of and disputes concerning being able to execute code, visualize programming concepts and interact with lecturers are explored along with relevant examples. The success and limitations of automated assessment are examined by looking at existing implementations. Finally, existing implementations of E-Learning systems are analyzed and compared. This paper concludes that E-Learning is an effective medium through which to teach programming, but that systems must be carefully developed to prevent wasting the opportunities provided by this new medium.

CCS Concepts

•Applied computing → Interactive learning environments; E-learning; Interactive learning environments; E-learning;

Keywords

Python; Web-Based; Tutor

1. INTRODUCTION

Computer Science education is facing a retention crisis [27]. Large numbers of students enroll in programming courses, but very few successfully complete them. High failure rates in programming courses can, in part, be attributed to difficulty adapting to an algorithmic way of thinking and the challenge of moving from the use of natural language to programming languages and students [2]. Programming is a vital skill that is constantly in demand. If high failure rates cannot be addressed, shortages of skilled professionals in the Computer Science field will only get worse. Learning to program requires time intensive practice that cannot feasibly be completed solely in a lecture or tutorial setting [24]. All students must practice programming in their own time if they wish to succeed. The introduction of a comprehensive E-Learning system can provide a useful platform for students to get through work on their own. Furthermore, interactive E-Learning can provide dynamic experiences that are impossible on a page.

In the next section, the effectiveness of E-learning as a teaching medium is discussed. The common components of E-Learning systems are discussed with relevant examples of implementations and automated marking is explored. Section 3 examines existing examples of E-Learning systems that are used to teach Python Programming. This section focuses specifically on Python education, as it is the intended target of this honours project. The selection of Python is briefly justified and various types of e-learning systems are examined. Section 4 discusses the differences and similarities between existing systems. Finally, Section 5 concludes the literature review.

2. E-LEARNING

E-learning is broadly defined as any instruction that is delivered electronically [14]. It has become increasingly more popular as the prevalence of Internet access and mobile devices has grown [1]. The appeal of E-Learning lies in its ability to allow students to engage actively in learning materials at their own pace, rather than merely receiving information in a typical classroom setting where they have little to no control over the speed at which information is received [32].

E-Learning has become somewhat of a hyped phenomenon over the last decade. It is important to separate the hype from reality to meaningfully assess the effectiveness of e-learning [47]. It is generally agreed that students can learn content that is delivered by technology [38, 4, 37]. However, the effectiveness of e-learning has been found to be conditional on the e-learning initiatives being well-focused and relatively small scale [33]. It has also been observed that students that are unfamiliar with or anxious about the use of computers perform worse off in e-learning initiatives than students that are comfortable with computers [30].

A literature review of 96 research reports, with data cumulatively gathered from 19331 subjects, found that e-learning initiatives were 6% more effective than traditional classroom instruction [43]. The benefits of e-learning include allowing students to move at their own pace, ability to track learner

DOI: 10.1145/1235

progress and the capacity to educate many students quickly [47]. Some studies surveyed found that there was no significant difference between the effectiveness of e-learning and classroom based instruction so it is important to understand that e-learning can be more effective than classroom instruction, but this is not always the case [47]. Educators should meaningfully compare the benefits of an e-learning initiative to the effort of implementing it and the investment in IT services and staff needed before committing to an e-learning venture.

Evidence collected suggests that the combination of e-learning and traditional classroom instruction can be highly effective [43]. Further research into this field is required before any definitive judgements can be made. Moore and Kearsley (2011) argue that instead of pitting one method against the other, it would be more productive to determine which students and topics are most suitable to each method, as the difference in effectiveness between the two appears to be limited [35].

2.1 Components of E-Learning Systems

This section describes the components that have commonly been used in E-Learning systems for introductory programming courses. The effectiveness of these components and existing implementations are discussed.

2.1.1 Interactive Code Examples

When beginning to learn how to program, it is important that students are able to focus on developing an understanding of computing concepts [13]. Traditional IDEs are designed with complex features to cater for experienced programmers. These features can be overwhelming for beginners and can very easily detract from the learning process [13]. In addition, installation and setup of these IDEs can be a frustrating barrier to learners; often, a large amount of time is spent on installing, configuring and understanding IDEs rather than on the actual coding. Simple, web-based execution of code snippets has been found to be a much more beginner-friendly way of introducing a student to programming. In the context of web-based systems, interactive code comes in the form of a notepad style frame, which may contain code pre-loaded by instructors, that students can execute and, in some cases, debug. This format is more suitable because web-browsers automatically download the necessary resources to execute code which appears in a not-threatening, familiar notepad style frame [45].

Cleveland University ran an experiment in their introductory course to C programming to determine the effect of an interactive web-based learning tool learning about pointers in C [32]. This tool contained interactive code frames which students could run and edit in their own time. In the first semester, the course was taught in a traditional classroom style. The basic concept of pointers was explained in lectures and examples were discussed in class. In the second semester, the traditional classroom approach was supplemented with an interactive learning tool. Again, basic pointer concepts were explained in class but this group was told to use the interactive learning tool to go through examples. At the end of each course, each group of students wrote a test on pointers which was a variation on the examples that has been presented in class and online respectively. The results were remarkably different for the two groups. Only 10% of the class with access to the online tool failed and 80% achieved marks over 90%; in the regular class, 30% failed and only 49% achieved over 90%. These results have largely been attributed to learners being able to work through the examples in their own time, rather than being limited to the time that a lecture allows. [32] It is, however, important to note that the more successful group participated in the study in the latter half of the academic year, so may have benefited from other programming experience.

Concerns about experiment design aside, the experience at Cleavland University makes a strong case for the benefits of interactive coding examples. This is consistent with findings that interactive learning environments promote deep cognitive processed when they engage learners in active learning [23] Active learning is when students are called on to apply information that they have learned, rather than just passively observing [7]. The use of interactivity is still a debatable issue as some studies have found that interactivity obstructs learning by unnecessarily increasing students' cognitive load [23]. The emerging consensus appears to be is that the degree to which students are required to engage with interactive tools determines the benefit that they confer.

2.1.1.1 Skulpt

Skulpt is a web-based Python interpreter that allows code to be executed entirely on the client side [13]. It is used by may e-learning systems as the basis for their interactive code sections. This is achieved by compiling the user's Python code to JavaScript at runtime and executing it in the browser's JavaScript virtual machine [34]. Skulpt consists of the typical components of a compiler: code is broken up into tokens by a lexer, a parser builds an abstract syntax tree, the semantic analyzer checks for semantic errors and a code generator produces runnable JavaScript code [45]. One of the greatest advantages of Skulpt is that it is entirely based on the client side. A server is not needed so potential problems of server-side congestion are avoided [45]. Its major drawback is that it does not provide debugging facilities of any kind. Skulpt is openly available and has successfully been adapted by multiple E-Learning endeavors [48].

2.1.2 Visualization of Code

Many concepts in computer science are inherently abstract. Visualization of these concepts is widely perceived to be effective in helping learners build conceptual models [36]. This belief cannot be uniformly supported by research, as there is still a degree of controversy surrounding the topic[18]. Clarke (1983) argues that the interactive way in which media is delivered to students produces educational benefits rather than the use of media itself [5]. Kozma (1994) disagrees, stating that Clarke's research has failed to consider the interaction between media and a learner's cognitive resources [26]. Cobb (1997) adds that Clarke's outright denial of the usefulness of media in education has stunted growth in what could have been a more thoroughly studied field [6]. This insight is particularly relevant with the introduction of the dynamic Web 2.0, which is radically different to the Web as Clarke and Kozma experienced decades ago. Hundhausen's (2002) more recent findings have indicated that, as with interactive code, students who engage with visualizations actively obtain better results than students who passively view visualizations [19]. Building a conceptual model of a how a programming language works is essential to successful programming [18]. According to construc-

tivist learning theory, computer science students do this by actively acquiring knowledge through the merging of what they observe and the models that they've already internalized rather than by rote learning of textbooks or lecture notes [3]. It has been shown that animation and graphics can be used to help students develop this internal model [41]. There are 4 different styles of visualization used in computer science education: program visualization, algorithm visualization, visual programming environments and microworlds [18].

2.1.2.1 Program Visualization

Program Visualization is the low level visualization of a program's source code [18]. This kind of visualization is most successful when attempting to debug programs as it provides visualizations of the program's current state (variable values, the content of the stack etc).

Online Python Tutor is a web-based program visualization tool that allows students to step through their code and debug programs with the aid of visualizations that represent the state of the program [13]. As it is web based, it requires no installation on the part of the learner. Users' code is passed to a backend server which uses the Python debugging module record the program's runtime state (the contents of the stack and heap) after each line of code is executed [45]. The program trace is the encoded in JSON format by adding additional metadata to the Python data types and serializing them into JSON types. Finally, the JSON data is visualized using the D3 and jsPlumb JavaScript libraries [15]. Online Python Tutor is severely limited by its use of a backend server. Programs are limited to 300 lines to prevent a bottleneck on the server side [45]. This may not be problematic for the short snippets of code that beginners deal with, but it raises issues of scalability. Additionally, a considerable amount of redundant data is stored by Online Python tutor because it saves the contents of the stack and heap after executing each line [15]. On a small scale, Online Python Tutor is an effective tool that beginners can use to visually debug their programs.

2.1.2.2 Algorithm Visualization

Algorithm Visualization provides a more conceptual visualization of the program than program visualization [18]. This high level visualization is used to explain the operation of algorithms. These systems are mostly used as teaching aids and require lecturers to add visualization code to algorithms or write animation code from scratch [18].

ALVIE, Algorithm Visualization Environment, was developed by Crescenzi and Nocentini (2007) at the University of Florence for use in a second year computer science course [10]. Algorithms are visualized after they have been executed using the interesting event approach. This means that visualization code is added to the algorithm when important steps in the process occur [10]. ALVIE was implemented in Java and is therefore highly portable. The effectiveness of AVLIE was tested in a class of 66 second year students, 43 of whom successfully completed the course [10]. In an end of course survey completed by the whole class, the material used received 8.04/10 for value [10]. In addition to the standard university survey, 20 students completed a survey specifically concerned with the use of ALVIE. When asked if ALVIE's visualizations helped them understand how the algorithms discussed in class worked, 19/20 students indicated that their experience was positive or very positive [10]. ALIVE's success has ensured its continued use at the University of Florence.

2.1.2.3 Visual Programming Environments

Visual Programming Environments provide users with building blocks containing code or pseudo code to create programs [18]. This kind of visualization allows students to begin to think algorithmically without having to understand complex syntax or programming environments [8].

Alice is a visual programming environment developed at Carnegie Mellon University [25]. It uses visual 3D objects such as cars or animlas alongside a drop and drag set of instructions to teach students to think algorithmically [25]. When Alice programs are executed, students can watch the effects of their instructions and are provided with pseudo code to familiarize them with the way in which actual programs are structures [25]. Alice was successfully used at Ithica College in a summer course aimed at preparing high school students for programming at university, with 85% of students indicating that their confidence in programming was 9 or 10 out of 10 [8].

California Lutheran University also used Alice as a learning tool in two basic (pre-CS1) programming courses: a regular day course for students and a nightschool course for adults [25]. Students indicated that they found the interactive animated environment helpful in understanding complex problems but those who moved on to CS1 struggled to move from drop-and-drag programming to the use of proper syntax and debugging [25]. It was also found that learning in Alice was very environment specific; students with poor logical thinking skills tended to struggle to solve problems outside of Alice. The experience at CLU indicates that while Alice may be effective in courses intended to prepare students for first year programming, the simplicity of a visual programming environment can fail to prepare students for actual programming problems.

2.1.2.4 Microworlds

Microworlds facilitate programming by providing a specialized teaching language which simplifies the number of instructions available to programmers [18]. Physical metaphors such a manipulating a robot are used to visualize program execution. Northwest Missouri State University tested the use of a microworld called Jeroo in their introductory Java programming course [11]. Jeroo is a kangaroo inspired animal that can move around an island in search of food while avoiding poachers' nets. Students are provided with very basic instructions (left, right, ahead, here) which are used to navigate the island map [11]. Basic methods such as isNet() and isFacing() are also provided. Over 2 seminars of use, 97 students were asked to complete a survey on the use of Jeroo [11]. On a likert scale (1-5), students indicated that Jeroo was effective in introducing them to programming, rating it 4.04/5 on average [11].

2.1.3 Instructor Feedback

Personal interaction in lectures and tutorials has been proven to be more effective than textbooks and example programs [7]. This is problematic because introductory Computer Science classes are often very large and must cover a substantial syllabus in a limited amount of time; lecturers often have to discourage questions in class, or limit the time they spend answering them for the sake of covering the material allocated to the time. Lecturers play a vital role in the use of e-learning in courses [29]. It has been found that web based learning is 19% more effective when students receive

feedback throughout the course [43].

2.2 Automated Assessment

2.2.1 Benefits of Automated Assessment

Before discussing automated assessment, it is important to note that no grading system can replace interaction with a real teacher [40]. This does not mean that these systems are without merit; teachers and students can benefit from automated assessment systems. Automated assessment systems can replace the marking process completely. The success of this approach is dependent on a capable plagiarism detection mechanism to prevent the students from taking advantage of the system [39]. Alternately, automated assessment tools can be used by teacher to analyze program features for large groups of students or track the progress of a particular student.

Automated assessment allows students to receive instant feedback on their work which can be helpful when tackling problems without a teacher present. This is particularly useful for beginner programmers who often make trivial mistakes which are easy to fix when provided with basic guidance [17]. In introductory programming courses, the use of an automated assessment system has the potential to free lecturers and tutors to answer more complex issues instead of wasting time on fixing trivial errors. It is, however, important to place limits on use of automated assessment systems to prevent students from adopting a trail-and-error, brute force approach to programming [39]. Another benefit of automated assessment systems is that they offer objectivity when marking that is often unatainable when large classes' work is marked by more than one person [20].

Reinforcement theory states that students' motivation and task involvement is influenced by the promise of performance feedback. In short, if students expect to be evaluated, they become motivated to achieve good results [28]. Motivation to succeed is vitally important for beginner programmers as they will often face long periods of debugging before success is achieved. Automated assessment can aid in providing beginner programmers with goals that motivate them to keep going. An end of term survey of students in an introductory programming class conducted at MIT on student perceptions of the usefulness of their automated assessment tool CAT SOOP, discussed later, found that 40% of students rated it 5/5 and 32% rated it 4/5 on a likert [17].

2.2.2 Implementing Automated Assessment

The majority of automated assessment systems check programs for correctness using black box testing. This is a method of testing that compares the output of the submitted code to the output of a model solution for a given set of input [22]. This is an effective way to approach automated marking because a wide range of programs can be marked in this way. The major drawback of black box testing is that creating sufficient tests to check for correctness is time consuming for lecturers. The black box method is limited to checking program correctness, as tests cannot assess students programming style [12]. This can be specifically problematic in beginners' programs where students may produce the right output but do so in a clumsy way that does not incorporate the components the assignment was aimed to test (using multiple nested if statements instead of a switch statement, for example). Automated assessment tools cannot award marks for programming style or innovation, but they can provide markers with analytic information that helps them determine that the program has taken on the expected form (a count of the number of switch statements, for example, can be used to ensure that students have employed the concept). Other automated assessment tools use machine learning to produce hints for students, but these systems are beyond the scope of this paper [21].

MIT's CAT SOOP automated assessment tool uses black box testing to determine the correctness of students' programs [17]. Assessment occurs on a public web server so code is examined before execution to ensure that it does not contain any statements that could potentially harm the server or interfere with assessment [17]. To prevent infinite looping, tests are only allocated 2 seconds of runtime each. CAT SOOP provides students with a visualization of the program trace using a version Online Python Tutor that has been modified to better explain errors to students.

The ASSYST system developed by Jackson and Usher checks programs for correctness, efficiency, complexity and style. The standard black box approach is used to test correctness [20]. In addition to this, lecturers or tutors are can access the students code and test data to ensure that it is correct. The amount of CPU time spent executing the student's program is compared to the time that a model solution takes to determine efficiency. This can be problematic for beginner snippets of code because they run in a infinitesimally small amount of time, so a statement execution count is also used to compare the student's solution to the model answer [20]. Metrics are used to evaluate complexity and style, where style is considered to be the desirable characteristics of a program (comments, indentation etc) that cannot specifically be noted in the assignment brief [22].

In place of instructor feedback, which is often unattainable for logistical reasons, automated assessment systems can be used to provide users with feedback. While feedback from a lecturer or tutor is always preferable, automated feedback is the next best way to encourage learners to actively engage in debugging their programs. Beginner programmers are often not capable of tracing program execution and debugging tools run the risk of unnecessarily adding to the cognitive load faced [44]. Program traces can be used to show users how the input has been processed to produce the output. Visualization of program traces has been proven to be effective in enhancing program comprehension for larger applications [9]. These results were, however, obtained studying industrial sized applications so may not be applicable in the context of simple, beginner programs. The University of Duisburg-Essen's automated assessment system JACK was tested on 400 first year students to determine the effectiveness of trace visualization in introductory programming courses [44]. It was found that, assuming that students understand how their programs should work, traces can be an effective way of determining exactly where their program is going wrong [44]. This experience suggests that the use of trace visualization is helpful to beginner programmers.

3. INTERACTIVE PYTHON E-LEARNING

Before examining various existing e-learning initiatives, this section justifies the selection of Python as an introductory programming language.

3.1 The Use of Python

The selection of an appropriate language is vitally important to the design of a programming course. This decision has been identified as one of the six grand challenges in computer education [31]. Instructors generally choose between high level languages, as their similarity to natural language makes them much more intuitive for beginners than lower level languages. The main factors that influence choice of language in computer science education are industry relevance, availability of useful tools and departmental preference [39]. Based on these criteria, one would naturally come to consider Java, Python, C++ and C as strong candidates for selection.

Cognitive overload has been identified as one of the difficulties that students face when first attempting to learn to program [2]. This occurs because students have to understand rigid syntax and commands before they can start to think algorithmically [42]. Python is able to reduce this overload because it has relatively straightforward syntax; this allows students to start writing programs relatively quickly [42]. In contrast, Java, C++ and C have been criticized for verbose syntax that imposes too-high a cognitive load on beginners [39].

Python has recently become more popular in universities for introductory programming courses [15]. MIT Professor John Guttag (2013) states that this is due to the variety of useful libraries available and ease of explaining concepts without the language itself interfering in understanding [16]. Python's gradual learning curve is also an appealing feature when dealing with beginner programmers [45]. There is no standard curriculum or set of goals set out for introductory programming courses so programming language selection must therefore occur on a case by case basis according to the goals of the course [39].

3.2 Interactive Textbooks

3.2.1 How to Think Like a Computer Scientist

How to think like a computer scientist is an interactive textbook that was designed with the goal of providing students with the goals of encouraging students to interact with the execu code in the book, reduce the barriers to programming produced by traditional IDEs and engage students in multiple learning modalities [39]. The creators of the ebook developed CodeSkulptor, a combination of Skulpt and a visualization tool called Codelens, to allow students to execute code in the textbook and to visually debug their programs [34]. Embedded videos were supplied in each section to give a brief overview of the concepts being covered. In addition, feedback from the test run by an automatic assessment tool were available to students for certain pre-set exercises. The textbook also allowed students to save and reload work. This feature was greatly appreciated as students often move between home and lab machines [34].

Luther College studied the use of How to Think Like a Computer Scientist over a 5 week introductory programming course [34]. Usage data and a survey complected by 66 students were used to evaluate effectiveness. Use of the in-book executable code blocks was found to be consistently high, with 93% of use in the first week and 83% in the last week of the semester. In contrast, use of the visualization feature began at 70% but consistently fell to 14% by week 4, with a slight increase to 18% in the last week of the course [2]. Despite these strong trends, it was found that success correlated more strongly with use of the visualization tool. Correlation does not, however, imply causality. Determining whether students performed well because they used CodeLens or if only well performing students used Codelens requires further study [34].

3.3 Online Tutors

3.3.1 PiLet

PiLet is a unique web-based tool for teaching Python programming because it was designed to cater for students of all learning styles. The tool provides executable examples of code, videos explaining concepts, programming puzzles in the form of code blocks that must be rearranged and assessment in the form of randomized multiple choice questions [1]. In future work, developers hope to adapt the system to use analytics of student use to personalize their learning experience to fit their learning style [1].

3.3.2 CS Circles

CS Circles is a free programming website that aims to teach introductory programming skills an an accessible way [40]. Lessons contain embedded executable code blocks that students are encouraged to interact with. When necessary, students can send messages with their code automatically included to their lecturer. These help messages, along with all code students have produced, are stored as part of the student's progress report, which can be accessed by lecturers at any time. CS Circles does not have an anti-plagiarism check so lecturer access to student records must be relied on to determine authenticity. An automated assessment system that runs on a central server is used to check the correctness of student submissions. Code can be debugged visually and pop up hints are available to help struggling students [40].

3.4 Massive Open Online Courses

3.4.1 Rice University MOOC

A MOOC called "An Introduction to Interactive Programming in Python" was developed at Rice University to help familiarize beginner programmers with programming concepts and techniques [46]. The MOOC was highly successful and is one of the top rated programming MOOCs available [45]. The MOOC uses Skulpt to execute code and Online Python Tutor to provide program visualization for debugging [45]. To prevent server side congestion, code and visualization data are generated on the client side [45].

4. DISCUSSION

Table 1 summarizes the features of the existing implementations discussed in section 3. Although it was not discussed specifically as a feature of e-learning, systems management of students work was a common and useful feature so it has been included in this comparison. System storage of student work is helpful for all students as they frequently move between university and home workstations. It is particularly helpful for beginner students who are less used to looking after their own work; they are far more likely to accidentally lose flash drives or let versions on different workstations out of sync with each other.

All of the existing implementations allow for interactive execution of code and visualization of the code. CS Circles is the only implementation that uses a visual programming

Figure 1: Table 1: Features of Existing Implementations

	Interactive Code	Visualization	Instructor Feedback	Automated Assessment	Store Student Work
How To think like a computer scientist	√	√	X	X	√
PyThy	√	√	X	√	√
PiLet	√	√	X	X	X
CS Circles	√	√	√	√	√
Rice University MOOC	√	√	X	X	√

environment. The use of scrambled blocks of code which must be put in the right order to create a program can be very beneficial for beginners because it allows them to focus on thinking algorithmically rather than focusing on syntax. The other implementations all use program visualization. This is helpful for beginner students who struggle to work through their programs mentally, but lack of interaction with the visualization limits effectiveness.

CS Circles is also the only implementation to offer interaction with a lecturer or tutor. This interaction has been found to be critical to learning how to program [7]. In some cases, specifically Rice University's MOOC, the scale of the project makes personal interaction with every student unfeasible. The next best solution to this would be to have hints available to students to at least mimic the effect of a lecturer or tutor. Automated assessment is only implemented in CS Circles and PyThy. This is to be expected because black box testing of programming assignments requires time-consuming specification of assignments and appropriate tests.

All of the implementations discussed offer students a valuable learning experience. CS Circles stands out as the implementation that has made the most effective use of the technology available to deliver a well-rounded e-learning experience. As discussed earlier, web-based learning is not guaranteed to be more effective than classroom based learning. It is therefore vital that e-learning systems justify the time and expense of their set up by taking full advantage of the features that only e-learning systems can offer.

5. CONCLUSIONS

The successful education of future generations of computer scientists is essential to the future growth of our discipline. The amount of practise required for of beginner programmers to be successful renders traditional classroom settings insufficient due to the limited time available. E-Learning provides a medium that can facilitate interactive, engaging out of classroom learning. However, e-learning systems are not guaranteed to be more effective than classroom settings. To justify the cost and resources required to set up an e-learning system, it must take full advantage of features such as interactive code, visualization and automated marking to actively engage students in the learning process. There is some evidence that the combination of e-learning and traditional classroom learning can be effective, but this hypothesis has not been explicitly tested. Development of an effective e-learning system for beginner programmers has the potential to aid in successful completion of programming courses.

6. REFERENCES

[1] B. Alshaigy, S. Kamal, F. Mitchell, C. Martin, and A. Aldea. Pilet: an interactive learning tool to teach python. In *Proceedings of the Workshop in Primary and Secondary Computing Education*, pages 76–79. ACM, London, UK, 2015.

[2] C. Alvarado, B. Morrison, B. Ericson, M. Guzdial, B. Miller, and D. L. Ranum. Performance and use evaluation of an electronic book for introductory python programming. 2012.

[3] M. Ben-Ari. Constructivism in computer science education. In *Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '98, pages 257–261, New York, NY, USA, 1998. ACM, New York, NY, USA.

[4] K. G. Brown. Using computers to deliver training: Which employees learn and why? *Personnel Psychology*, 54(2):271–296, 2001.

[5] R. E. Clark. Reconsidering research on learning from media. *Review of educational research*, 53(4):445–459, 1983.

[6] T. Cobb. Cognitive efficiency: Toward a revised theory of media. *Educational Technology Research and Development*, 45(4):21–35, 1997.

[7] D. A. Cook and D. M. Dupras. A practical guide to developing effective web-based learning. *Journal of General Internal Medicine*, 19(6):698–707, 2004.

[8] S. Cooper, W. Dann, and R. Pausch. Alice: a 3-d tool for introductory programming concepts. In *Journal of Computing Sciences in Colleges*, volume 15, pages 107–116. Consortium for Computing Sciences in Colleges, 2000.

[9] B. Cornelissen, A. Zaidman, and A. Van Deursen. A controlled experiment for program comprehension through trace visualization. *Software Engineering, IEEE Transactions on*, 37(3):341–355, 2011.

[10] P. Crescenzi and C. Nocentini. Fully integrating algorithm visualization into a cs2 course.: A two-year experience. In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '07, pages 296–300, New York, NY, USA, 2007. ACM, New York, NY, USA.

[11] B. Dorn and D. Sanders. Using jeroo to introduce object-oriented programming. In *Frontiers in Education, 2003. FIE 2003 33rd Annual*, volume 1, pages T4C–22. IEEE, 2003.

[12] C. Douce, D. Livingstone, and J. Orwell. Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput.*, 5(3), Sept. 2005.

[13] S. H. Edwards, D. S. Tilden, and A. Allevato. Pythy: Improving the introductory python programming experience. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 641–646, New York, NY, USA, 2014. ACM, Blacksburg USA.

[14] T. Govindasamy. Successful implementation of e-learning: Pedagogical considerations. *The internet and higher education*, 4(3):287–299, 2001.

[15] P. J. Guo. Online python tutor: Embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM Technical Symposium on*

Computer Science Education, SIGCSE '13, pages 579–584, New York, NY, USA, 2013. ACM.

[16] J. V. Guttag. *Introduction to Computation and Programming Using Python*. Mit Press, 2013.

[17] A. J. Hartz. *CAT-SOOP: A tool for automatic collection and assessment of homework exercises*. PhD thesis, Massachusetts Institute of Technology, 2012.

[18] J. Helminen and L. Malmi. Jype-a program visualization and programming exercise tool for python. In *Proceedings of the 5th international symposium on Software visualization*, pages 153–162. ACM, New York, NY, USA, 2010.

[19] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3):259–290, 2002.

[20] D. Jackson and M. Usher. Grading student programs using assyst. In *ACM SIGCSE Bulletin*, volume 29, pages 335–339. ACM, California, USA, 1997.

[21] W. Jin, L. Lehmann, M. Johnson, M. Eagle, B. Mostafavi, T. Barnes, and J. Stamper. Towards automatic hint generation for a data-driven novice programming tutor. In *Workshop on Knowledge Discovery in Educational Data, 17th ACM Conference on Knowledge Discovery and Data Mining*. Citeseer, 2011.

[22] M. Joy, N. Griffiths, and R. Boyatt. The boss online submission and assessment system. *J. Educ. Resour. Comput.*, 5(3), Sept. 2005.

[23] S. Kalyuga. Enhancing instructional efficiency of interactive e-learning environments: A cognitive load perspective. *Educational Psychology Review*, 19(3):387–399, 2007.

[24] C. Kelleher and R. Pausch. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, 37(2):83–137, 2005.

[25] M. Klassen. Visual approach for teaching programming concepts. In *Proceedings of the 9th International Conference on Engineering Education (ICEE 2006)*, pages 23–28, 2006.

[26] R. B. Kozma. Will media influence learning? reframing the debate. *Educational technology research and development*, 42(2):7–19, 1994.

[27] E. Lahtinen, K. Ala-Mutka, and H. M. Järvinen. A study of the difficulties of novice programmers. In *ACM SIGCSE Bulletin*, volume 37, pages 14–18. ACM, 2005.

[28] K. M. Y. Law, V. C. S. Lee, and Y. T. Yu. Learning motivation in e-learning facilitated computer programming courses. *Computers Education*, 55(1):218–228, 8 2010.

[29] S. Liu. Faculty use of technologies in online courses. 2005.

[30] J. J. Martocchio. Effects of conceptions of ability on anxiety, self-efficacy, and learning in training. *Journal of applied psychology*, 79(6):819, 1994.

[31] A. D. McGettrick. *Grand challenges in computing: Education*. British Computer Society, 2004.

[32] D. R. McIntyre and F. G. Wolff. An experiment with www interactive learning in university education. *Computers Education*, 31(3):255–264, 11 1998.

[33] R. Michaelson. Does e-learning work? 2002.

[34] B. Miller and D. Ranum. Beyond pdf and epub: Toward an interactive textbook. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 150–155, New York, NY, USA, 2012. ACM, New York, NY, USA.

[35] M. G. Moore and G. Kearsley. *Distance education: A systems view of online learning*. Cengage Learning, 2011.

[36] T. L. Naps, G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, et al. Exploring the role of visualization and engagement in computer science education. In *ACM Sigcse Bulletin*, volume 35, pages 131–152. ACM, New York, NY, USA, 2002.

[37] R. F. J. North, D. M. Strain, and L. Abbott. Training teachers in computer-based management information systems. *Journal of Computer Assisted Learning*, 16(1):27–40, 2000.

[38] J. M. O'Hara. The retention of skills acquired through simulator-based training. *Ergonomics*, 33(9):1143–1153, 1990.

[39] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson. A survey of literature on the teaching of introductory programming. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '07, pages 204–223, New York, NY, USA, 2007. ACM, New York, NY, USA.

[40] D. Pritchard and T. Vasiga. Cs circles: An in-browser python course for beginners. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 591–596, New York, NY, USA, 2013. ACM, New York, NY, USA.

[41] S. H. Rodger. Introducing computer science through animation and virtual worlds. In *ACM SIGCSE Bulletin*, volume 34, pages 186–190. ACM, New York, NY, USA, 2002.

[42] E. Shein. Python for beginners. *Communications of the ACM*, 58(3):19–21, 2015.

[43] T. Sitzmann, K. Kraiger, D. Stewart, and R. Wisher. The comparative effectiveness of web-based and classroom instruction: A meta-analysis. *Personnel psychology*, 59(3):623–664, 2006.

[44] M. Striewe and M. Goedicke. Using run time traces in automated programming tutoring. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 303–307, New York, NY, USA, 2011. ACM, New York, NY, USA.

[45] L. Tang. A browser-based program execution visualizer for learning interactive programming in python, 2015.

[46] T. Tang, S. Rixner, and J. Warren. An environment for learning interactive programming. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 671–676. ACM, New York, NY, USA, 2014.

[47] E. T. Welsh, C. R. Wanberg, K. G. Brown, and M. J. Simmering. E-learning: emerging uses, empirical

results and future directions. *International Journal of Training and Development*, 7(4):245–258, 2003.

[48] www.skulpt.com.