Visualization and Interactivity Methods to Improve the First-Year Computer Science Learning Environment

Literature Review

Siviwe Qolohle
Department of Computer Science
University of Cape Town
Cape Town Western Cape
qlhsiv001@myuct.ac.za

ABSTRACT

Learning how to program is often a daunting task for first-year university students with no prior coding experience. The skill of coding often viewed as too abstract to grasp for first, hence discouraging them from continuing with the course. With the demand for individuals with programming skills in the workplace increasing, it is urgent for the negative mindset associated with computer science to be corrected. This literature review explores different programming learning environments that are beginner friendly. It examines how visualization and interactivity are used to enhance the learning environment for first years and simplify coding concepts. Between Interactivity and Visualisation, the paper finds that Interactivity and Visualisation go hand in hand, with the feature of Interactivity is more crucial for first-year students. The review finds that previously created system may be too complex for first years and hence, investigates ways to adapt previous systems to make learning easier for first years.

CCS CONCEPTS

• Computer Science • Debugging • Programming

KEYWORDS

Programming, Visualisation, Teaching, Interactivity

1 INTRODUCTION AND MOTIVATION

One of the courses that make a first-year's transition from high school to varsity even more challenging is Computer Science. Many students are not taught to program in high school. Introductory Programming therefore has a critical role to play, to ease first-year students into Computer Science, without overwhelming them whilst developing a firm programming foundation at the same time.

The Demand for programming in the labour market is increasing [12]. Programming skills are not only required in degrees related to computer science, but they are also required in many other disciplines [11]. There is, however, a shortage of programmers because students have a preconceived idea of programming being difficult and hence do not want bother learning the skill [12]. With students doing introductory programming courses, their failure to understand the courses can be seen by the failure of students to grasp the fundamental skills of programming [11]. There are high failure rates within the introductory programming courses [5]. Computer Science is described as a subject that really challenges the mind and requires the mastery of problem-solving skills [2]. Many first years arrive at university without having mastered or efficiently exercised the skill of problem solving [2, 9]. This is an issue seeing that problem solving makes up the foundation of programming. Incorporating more visualization and interactivity in the teaching of computer has been proven to enhance the learning experience for students and improve their understanding of the basics of Computer Science [12].

This literature review aims to provide an overview of why first-year students struggle when learning the basics of programming and it aims to discover effective teaching methods which can be used for first-year students to ease them into the subject of computer science. The literature review also aims to provide an overview of systems which have already been implemented that make use of visualization and interactivity to improve understanding within introductory programming courses. These systems will also be adapted to check and enhance the problem-solving skills of first-year computer science students.

2 LEARNING COMPUTER SCIENCE

2.1 Challenges

2.1.1 Problem Solving Skills. Programming requires logical skills and computational skills [2]. The major goal of first year programming courses is to develop and a student's problem-solving skills and for them to learn a programming language that allows them to code the solutions to the problems that they have just solved [2]. The Programming requires in-depth problem-solving skills that many first-year students may not have acquired yet [2]. When students arrive at university their problem-solving skills do not improve quick enough because when lecturers teach, they provide solutions to the problems they give students instead of equipping them with the problem-solving skills to solve all

problems of that kind [18]. Spoon-feeding students answers during class does not help students master the basics which give them the ability to solve problems on their own. When teaching programming to first-years problem should be classified into types and solutions to the type of problem should be specified. For example, classifying the types of problems requiring 'while loops' or 'if-statements". This is, however, only applicable to introductory programming courses because solutions to more difficult computer science courses combine the above stated solutions into one programming consisting of 'while loops', if-statements and 'for loops'.

2.1.2 Lack of Assistance. For a first-year student with no prior coding experience, not having easy access to assistance is also a problem. Struggling with a problem and needing to wait for a long time for assistance from an educator can be demotivating for a student. The problem the student faces appears as an obstacle they cannot pass due to lack of assistance [12]. This is even more likely for a student that has no background on a subject and does not even know where to start when looking for supplementary resources (such as videos) for assistance.

2.1.3 Grasping the Basics. It is difficult to initially get an idea of what programming is about because it is so different to other subjects students learn in high school. In this case individualized learning is ideal but is not practical due to the large of students learning how to program [2]. Learning a programming language itself is also challenging as each language comes with its own syntax rules [5]. This means that on top of improving problem solving skills and using code to solve those problems, a language must also be learned to deliver that solution. Students are also often required to make use of content from other subjects when coming up with solutions to raised problems. This content can include mathematical and scientific formulae [2]. Gasparintatou and Grigoriadou [1] state that it has been discovered that providing students who have little to no experience in programming with highly cohesive text. Text that is highly cohesive and hence more comprehensive help sets a firmer foundation for students for the content they are learning.

2.1.4 Lack of Engagement. Learning how to code requires constant engagement [2]. Taking into account that first years have a lot courses that they are doing at once, constant engagement is a difficult requirement to meet. Making the application fun would assist in ensuring that students are working on improving their skills in their own time. A Research was conducted which tested students' academic performance after they made use of the game that was created to teach them computer science. The results indicated that incorporating entertainment into the learning experience improves students' academic performance [16].

2.2 Effective Teaching Methods for Programming

One of the new methods which are being used to teach students is Live-Coding [4]. This form of teaching consists of a teacher writing code in front of students and allowing students to ask questions as they write the code [12]. It is preferred by students as it allows students to be more involved in the learning process, seeing that they are able to stop the teacher at any time and ask them a question whilst they are writing the code [4]. Students are also able to get a glimpse into the teachers thought process and learn from their teacher's mistakes [4]. When devising a solution to a problem using programming, educators should be intentional about showing students their thought process (rather than making use of slides that already have solutions on them) [3]. This allows a student to apply their educator's thinking pattern when faced with a similar problem [3]. This can also allow the student to categorize certain solutions to certain types of problems.

From a study conducted with students who were learning computer science, they were asked about which teaching methods they think are most effective for computer science [9]. Laboratory practice and projects were placed first and second respectively, with lectures being placed third. laboratory practice allows students to put their lessons in effect to test whether or not they understood the content they were taught [9]. Laboratory practice also allows students to get the out from their input immediately, which assists in speeding up the learning process. Projects assign tasks to the students [9]. The students believed that the projects ensure that students have a proper understanding of the content and they stated that it assisted them in mastering the required computer science skills. Projects and laboratory practices can hence combat the effects of lack of Engagement.

3 VISUALISATION

Visualisation plays a critical role in understanding and teaching programming. It helps students graphically understand how a program works) [15]. Visualization helps the student visualize variables as well as how those variables interact with other variables [7]. Visualisation also helps the user see how objects are affected by sequences of instructions [7]. With visualization we can see how an algorithm works graphically [3]. Seeing that interactivity and visualisation often go hand-in-hand, in this review visualization refers to a student viewing content, and being able to have little (pressing next as the code is executed line by line) to no interaction.

3.1 Systems Using Visualisation

3.1.1 Improv. Live-coding is a clear example of incorporating visualization into the teaching process, in computer science. Live-coding led to the development of Improv[4]. In this application, the teacher writes and tests code in any language using an IDE. Improv is used to attach shortcuts to the IDE which allow the terminal or block of code to be displayed in a PowerPoint-like style set of slides. This is all done live, meaning

as the educator is teaching and typing the code, it is being displayed in the above stated format. Components such as text and images can be appended to these slides. Studies have found that it is important to for students to see code being written and to see the effects of their 'what-if' questions [4]. Improv allows for multiple editable files to be open at once. On one slide, multiple editors can be edited. For example, the backend and front end of a code can be displayed on the same slide as well as the output to see how certain changes in the code of the backend and the frontend affect the output. The versatility of the application caters to different types of teaching methods in which educators would like to teach their programming content. The platform makes it easy for students to quickly grasp concepts [4].

After giving students the opportunity to learn using Improv it was found that the lack of required context switching resulted in a reduction of their cognitive load [4]. When it comes to coding, context switching (switching between different windows, or having multiple windows occupy different parts of the screen) can have a negative impact on the learning process and can increase the time it takes for one to grasp a concept. It affects concentration and can result in one reading over a block of code multiple times. Navigating through multiple windows makes it easy get lost and confused as a presenter, making it worse for the viewers [4]. Pre-made slides also require context switching as the lecturers usually need to open a new window the display what they have just discussed on the slides. Presenters also sometimes must switch between the IDE and the terminal to show the code as well as the output it provides from a given input. To visualize the output, context switching is usually required [4].

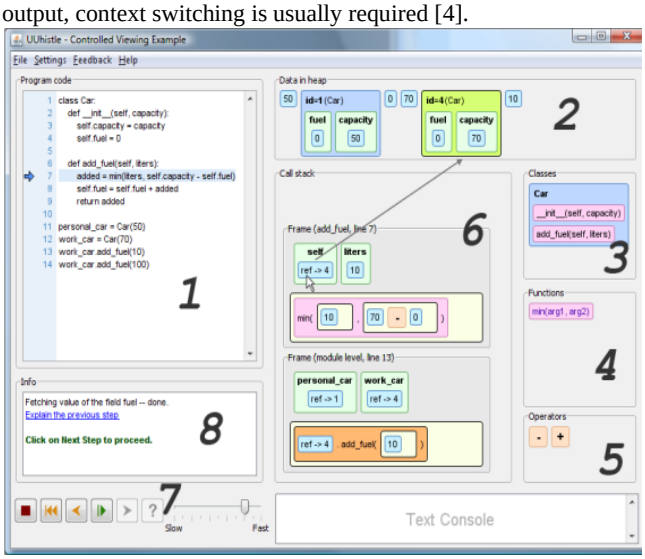

Figure 1: UUhisle Controlled Viewing mode.

3.1.2 UUhistle. UUhistle is a program written in Java, that has both a predominantly visual mode and interactive mode (see figure 1) [10]. The program can be run on its own or on the web. The predominantly visual mode is referred to as Controlled Viewing and it allows the user to observe as the program executes showing the different steps the program follows. The code is displayed on the left (with the current line highlighted). On the right side are the components which make up the program, these components include the classes, variables, functions, and operators. As each line is executed, the changes to variables are visualised as well as the steps taken when a method is called by a variable. The changes made by each transition to the next line are displayed on the right. In the Controlled Viewing state the user can also make use of buttons such as Stop, Rewind, Undo and Next Step as the lines execute.

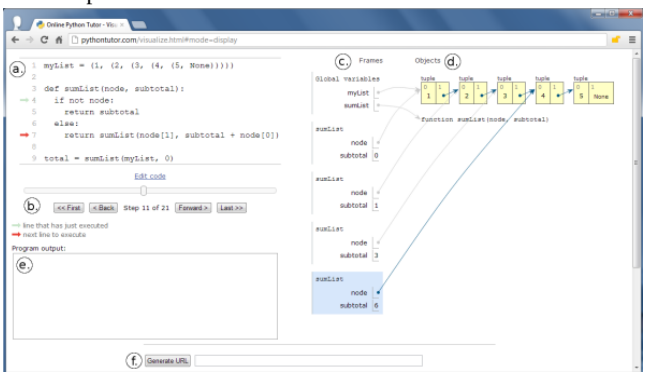

Figure 2: Online Python Tutor

3.1.3 Online Python Tutor. Online Python Tutor is a Python visualization program based on the web (see Figure 2 above) [13]. Students struggle to understand how a block of code links to a process that can cause multiple changes to an object [13]. The program can be seen as a simpler version of the Controlled Viewing mode of UUhistle. The program allows the user to step forwards and backwards through the code line by line. The interface displays the code, the output, and a visualization of each transition to a new line, all on the same page. The success of the program can be seen by the fact that the program is used in first-year Computer Science courses in many universities [13]. These universities include MIT, University of Washington, and UC Berkeley. In 2013 it was reported that over 30,000 individuals use Online Python Tutor per month [13].

4 INTERACTIVITY

Interactivity also plays a crucial role when learn how to code. It allows students to not only visualize their code as well as its effects, but it also allows the user to fully interact with it and gives students. Often interaction also includes giving students intrcutions to write code [12]. Past experiments which will be further elaborated on below actually prove that interactivity may be more effective than visualization.

4.1 Systems or experiments using interactivity.

An experiment was conducted determining the effects of an interactive programming application on the success rate of students in programming [12]. The application consisted of several activities. An example of an activity the application

would provide is giving the student a scenario where an individual is given a few objects (for example, 5 apples), and the student is then tasked to store the objects in a basket. The solution to this task should be provided using a line of code. the expected solution to this problem is: "int basket = 5" (if the program was being written in Java). This example teaches the student about the basics of variables. Other features are also included such as hovering over a line of code, and having the application display a description of the syntax of the highlighted area. In the experiment, students get taken through the process of developing functions and they are also given the opportunity to play games. Some games in the application consisted of puzzles where the students would the line of code in its correct position. To ensure user's skills are constantly improving, the users are taken through different levels where the code is appended to each time you reach a higher level. Real-life scenarios are used to give the student a clear idea of how the code should be created. This experiment assisted in enhancing critical thinking and problem-solving skills [12].

Two experiments were conducted mainly focusing on the importance of using interactivity when learning programming. One of the experiments conducted was to determine the impact of constructing visualizations of sorting algorithms on students' understanding and attitude towards programming [6]. They went in with some programming knowledge and they were also taught about graphics and animation design. The research mainly focused on determining the effect of viewing pre-made visualizations vs creating your own visualizations. The topics were centered around sorting algorithms. It was found that constructing your own algorithms improved students' performance. Interactivity increases students' understanding. Students indicated that constructing the algorithms themselves helped make the code more concrete for them.

The other experiment investigated the impact of sorting numbers using physical index cards [8]. The goal was to prove the effectiveness of low-tech solutions compared to simple code walk through. the usage of index cards produced a deeper understanding of the sorting algorithms. The sorting of digits using cards can be transformed into program by simply using a drag and drop feature.

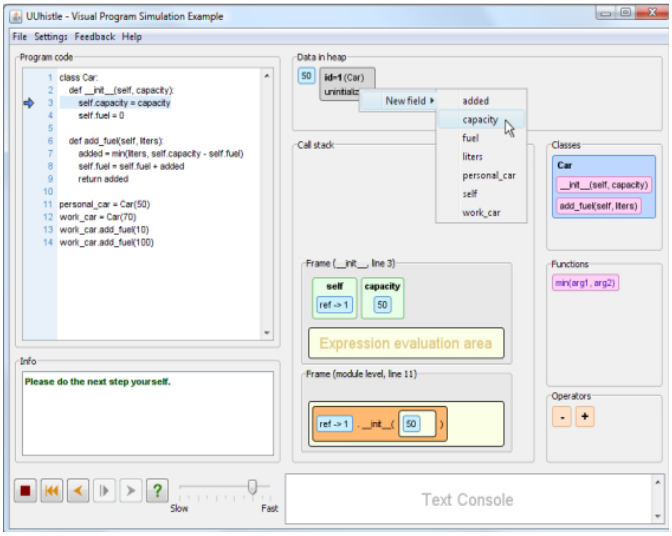

Figure 3: UUhistle VPS mode

The second mode of UUhistle is a Visual Program Simulation (VPS) (see figure 3) [10]. In this mode, the student takes on the role of the computer. The objective of this mode is to give the student a deeper understanding of what the computer does when a program is run. In other words, in the VPS mode, the student manually does the changes observed on the right-hand side of the Controlled Viewing (figure 1) mode in each line of code. When a method is created, the user is expected to store it in memory; when the method is called by a variable, it is the user's responsibility to drag the function and enter the required parameter (as with the Controlled Viewing mode, the stored method is displayed on the right side of the screen).

5 COMPREHENSIVE ERROR MESSAGES

When creating an educational programming application, a lot of time needs to be spent on determining how the application responds to errors made by the user. The ability to detect errors in code is crucial for beginner programmer for them to succeed programming. Error messages have the potential to greatly guide and assist first in learning how to code [14]. However, the error messages are currently too difficult to comprehend for beginner programmer [14].Python's error messages can be difficult to understand for a first-year programmer, for example, getting the following message: "Syntax Error" can be difficult to comprehend because a first-year may not even know what the word syntax means in programming. Some syntax errors may not even come with further explanation. Due to inefficient error message someone may struggle for hours to find why their program keeps giving them an error message only to find out that they misspelt their variable [17]. TigerJython is a python environment that makes use of more comprehensive error message [17]. The environment prints the error message in the shell as well as explains the error or makes a suggestion under the line of code that is giving the error. Students were asked to rate from 1 to 5

the effectiveness of TigerJython when it comes to error messages. The average rating was 3.3. The students were either high school students or first year beginner programmers. When examining the results of students using the program, it was found that a third of the errors experienced by the students were insignificant syntax errors.

UUhistle also allows students to make mistakes so they can learn from them. The program reacts to the error by suggesting that the user undo or fix the error [10].

6 DISCUSSION

Seeing that many students struggle to initially grasp the basics of computer science, this highlights that the starting point for the course is too high or advanced for students. It has also been found from past research that there is a lack in the problem-solving skills among students [2]. Grasping the basics of coding makes students more qualified to solve problems on their own, resulting in improved problem solving skills. To assist students, the gap must be bridged between high school and university. The goal of the educators should be focusing on using programming to improve problem solving skills. The goal should not only be to teach programming. It can be seen from previous articles that jumping straight into programming is not working for students. It was stated that when learning a new skill, beginners learn better from comprehensive text. This implies that prior to asking a student to write code, a comprehensive description of the topics the students will be making use of in that code should be explained. For example, prior to asking students to use an array in a code, an in-depth explanation of the purpose of arrays as well as how they can be used should be given. Although the systems which have been touched on above have proven to be effective when it comes to assisting students, they do not give descriptions of the topics within computer science. Attempting to incorporate lessons into an application which assists student with learning to program which includes interactivity could be too ambitious, however it would be possible to give brief description of a topic prior to asking students to write a program. Incorporating this could help students fully grasp the basics of programming.

From past experiments and learning systems which are in use, it can be observed that visualization and interactivity Often go hand in hand. Although this literature review separated visualisation and interactivity into two different categories, most often, visualisation (when it comes to programming), does not come without a little bit of interactivity. For example, stepping through a code line by line counts as a form of interactivity. Although visualisation is important, as it helps students graphically understand a program and also allows students to see how variables act with each other, being intentional about incorporating a lot of interactivity is crucial [6, 7, 8, 15]. This is highlighted by the experiment which outlined that writing a program is better than viewing one [6]. Another experiment which highlighted this was the one that outlined that when learning sorting, physically sorting through digits is better than observing a sorting algorithm [8]. The importance of the inclusion of interactivity is also highlight by the fact that from a survey asking students to state preferred learning methods, laboratory practices and projects were the top two methods [9]. It is evident though that from the systems focusing mainly on visualisation and those focusing mainly on interactivity, that having a feature that allows students to step through code line by line and see the effect of each line, is crucial.

When it comes to the visuals, a lot can be learned from UUhistle, Improv and Online Python Tutor. All the programs display all of their features on one interface in order to avoid context switching. As described above, avoiding context switching greatly benefits the learner. The code, the output of the code and additional descriptions and explanations are on one page. The Improv interface may however be too complicated for a first-year student meaning UUhistle and Online Python Tutor would have the ideal layout.

Constant engagement has been stated as being crucial when it comes to learning how to program [2]. Some papers did stress on the effectiveness of incorporating entertainment into the program to make it more appealing to the student, and hence, increase engagement [16]. Incorporating entertainment does however seem ambitious when the focus is interactivity and visualization. Focusing on making the design and layout more appealing to students can however assistance in attracting students to the application.

When creating a visualization and interactivity program allowing students to make errors is important. Having simple and cohesive error messages is also crucial [14, 17]. As touched on above, when a beginner programmer encounters an error in their code, they can become extremely discouraged because often they may not even know where to begin the debugging process. Receiving a complicated error message only makes things worse for the beginner. Therefore, providing additional assistance in the form of cohesive error messages as well as suggestions for the user can be extremely beneficial.

7 CONCLUSION

This literature review touched on the teaching of introductory Computer Science courses. It highlighted the issues which result in a large volume of students fearing and not coping in the courses. It highlighted how lack of problem-solving skills, lack of engagement and not having grasped the basics contributes to the high failure rate of first years Computer Science. It also touched on effective teaching methods that can be implemented to teach students. Such methods include making use of Live-Coding as well as projects and laboratory practical.

Although the collected literature does give an idea of how applications to teach students about the basics of computer

science, more can be incorporated. The additional features should focus more on comprehensively teaching basics. Along with improving programming skills this would also improve the problem-solving skills which many first-years are lacking. Focusing on improving basics gives students more independence in the sense that they do not have to depend on memorizing rules of regarding when a certain statement or function should be used. Having more of a solid foundation makes knowing which syntax to use more intuitive.

From the literature, it has been found that multiple features need to be included in an introductory programming application. One of the features are having information on one page to reduce context switching. This feature should obviously be implemented in such a way that the student is not too overwhelmed by the abundance of content. Another feature is appropriate error handling. This includes suggestions of how to fix errors as well as descriptions explaining the error. Other features are an appealing layout, visualization and a lot of interactivity. The final feature is an educative one that gives brief descriptions of programming rules.

REFERENCES

[1] Alexandra Gasparinatou and Maria Grigoriadou. 2011. Supporting students. Computer Science Education 21, 1 (2011), 1-28. DOI: https://doi.org/10.1080/08993408.2010.509909

[2] Anabela Gomes and António José Mendes. 2007. An environment to improve programming education. In Proceedings of the 2007 international conference on Computer systems and technologies (CompSysTech '07). Association for Computing Machinery, New York, NY, USA, Article 88, 1–6. https://doi-org.ezproxy.uct.ac.za/10.1145/1330598.1330691

[3] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. 2007. 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). Association for Computing Machinery, New York, NY, USA, 204–223. https://doi-org.ezproxy.uct.ac.za/10.1145/1345443.1345441

[4] Charles H. Chen and Philip J. Guo. 2019. Improv: Teaching Programming at Scale via Live Coding. In Proceedings of the Sixth (2019) ACM Conference on Learning @ Scale (L@S '19). Association for Computing Machinery, New York, NY, USA, Article 9, 1–10. https://doi-org.ezproxy.uct.ac.za/10.1145/3330430.3333627

[5] G. Silva-Maceda, P. David Arjona-Villicaña and F. Edgar Castillo-Barrera, "More Time or Better Tools? A Large-Scale Retrospective Comparison of Pedagogical Approaches to Teach Programming," in IEEE Transactions on Education, vol. 59, no. 4, pp. 274-281, Nov. 2016, doi: 10.1109/TE.2016.2535207

[6] Ibrahim Cetin & Christine Andrews-Larson (2016) Learning sorting algorithms through visualization construction, Computer Science Education, 26:1, 27-43, DOI: 10.1080/08993408.2016.1160664

[7] Imre BENDE. 2022. Data Visualization in Programming Education. Acta Didactica Napocensia 15, 1 (2022), 52-60. DOI: https://doi.org/10.24193/adn.15.1.5

[8] J Geller and R Dios 1998. A low-tech, hands-on approach to teaching sorting algorithms to working students, 89-103. https://doi-org.ezproxy.uct.ac.za/10.1016/S0360-1315(98)00021-9

[9] J. Kingsley Arthur and K. Sarpong Adu-Manu. Causes of Failure of Students in Computer Programming Courses: The Teacher Learner Perspective" in International Journal of Computer Applications 77(12):27-32, Sept. 2013, doi: 10.5120/13448-1311

[10] Juha Sorva and Teemu Sirkiä. 2010. UUhistle: a software tool for visual program simulation. In Proceedings of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling '10). Association for Computing Machinery, New York, NY, USA, 49–54. https://doi-org.ezproxy.uct.ac.za/10.1145/1930464.1930471

[11] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. SIGCSE Bull. 33, 4 (December 2001), 125–180. https://doi-org.ezproxy.uct.ac.za/10.1145/572139.57218

[12] Prasad, A. et al. (2022) Programming skills: Visualization, interaction, home language and problem solving. Education and information technologies. [Online] 27 (3), 3197–3223.

[13] Philip J. Guo. 2013. 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). Association for Computing Machinery, New York, NY, USA, 579–584. https://doi-org.ezproxy.uct.ac.za/10.1145/2445196.2445368

[14] Rachel D'souza, Mahima Bhayana, Marzieh Ahmadzadeh, and Brian Harrington. 2019. A Mixed-Methods Study of Novice Programmer Interaction with Python Error Messages. In Proceedings of the Western Canadian Conference on Computing Education (WCCCE '19). Association for Computing Machinery, New York, NY, USA, Article 15, 1–2. https://doi-org.ezproxy.uct.ac.za/10.1145/3314994.3325090

[15] Šimoňák, Slavomír..2014 "Using algorithm visualizations in computer science education" Open Computer Science, vol. 4, no. 3, pp. 183-190. https://doi.org/10.2478/s13537-014-0215-4

[16] Tsung-Yu Liu. 2014. Using educational games and simulation software in a computer science course: learning achievements and student flow experiences. Interactive Learning Environments 24, 4 (2014), 724-744. DOI:https://doi.org/10.1080/10494820.2014.917109

[17] Tobias Kohn and Bill Manaris. 2020. Tell Me What's Wrong: A Python IDE with Error Messages. In Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20). Association for Computing Machinery, New York, NY, USA, 1054–1060. https://doi.org/10.1145/3328778.3366920

[18] William M. Waite. 2006. The compiler course in today's curriculum: three strategies. SIGCSE Bull. 38, 1 (March 2006), 87–91. https://doi-org.ezproxy.uct.ac.za/10.1145/1124706.1121371