JSPromptSolver: A Learning Tool Designed For Novice Learners In An Introductory JavaScript Cohort

Eric P. Katz
ekatz33@gatech.edu

***Abstract*—** Research has shown that learning computer programming is considered a difficult task. The number of learning elements which need to be understood in parallel can often be frustrating and result in failure for novice learners. Coding tools, also known as Integrated Development Environments (IDE), can play a role in the learning outcomes based on how difficult they are to install as well as the complexity of their interfaces. The nature of programming assignments, which often have students attempting to write code before comprehending the subject matter, also have been shown to present problems for learners. Coding tools which address these issues are needed. We present an easy to administer supplemental tool which we believe will lead to better learning outcomes for use in cohorts which need to teach introductory JavaScript.

Final Presentation Video

1 Introduction

Learning to program is difficult. The complexity of the topic and the parallel set of skills needed leaves little margin for error (Jenkins 2002). This paper will discuss why it is difficult and how we can improve code practice tools by limiting distractions and providing more formative feedback. An overview of what features make an ideal novice programming environment, based on the research presented in this paper, is shown below.

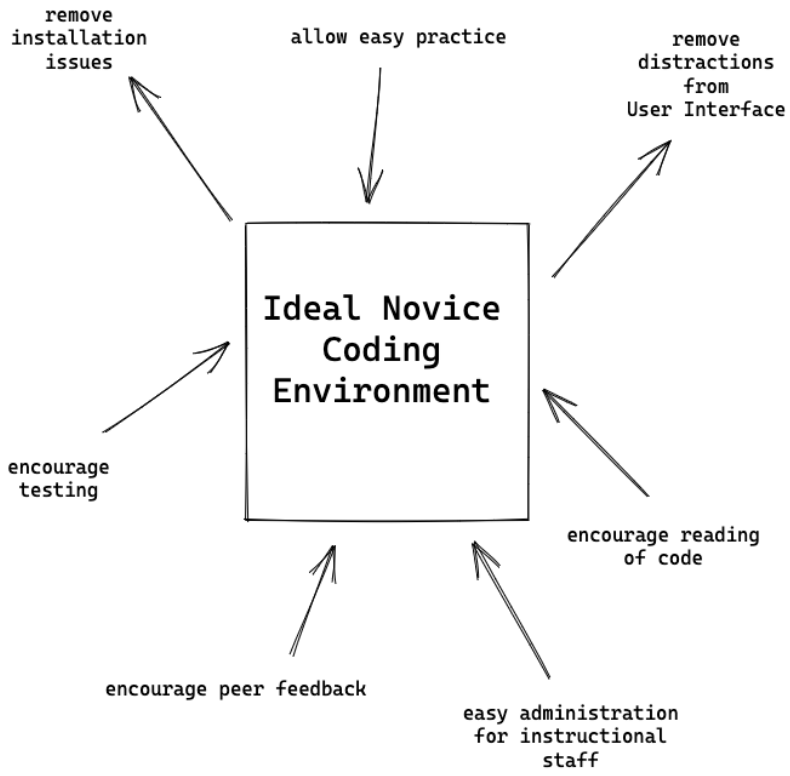

Figure 1: An Ideal Novice Coding Environment

- Part 2 of this paper will focus on research which explains the difficulty of programming in the context of cognitive overload.
- Part 3 of this paper will cover how extraneous cognitive overload in the form of coding environments, as well as installation issues, can hamper learning.
- Part 4 of this paper will focus on why conventional instructional material forces the user to learn by having them write code before understanding it.
- Part 5 will discuss ways we can improve learning outcomes by improving environment issues and providing formative feedback.
- Part 6 which will discuss available tools and why they don't meet the needs of novice learners.
- Part 7 will present JSPromptSolver, an easy to administer online tool which can be used in cohorts which need to teach students JavaScript.

2 Why Programming is difficult

As noted by Jenkins (2002), programmers must be able to do many things in order to be able write meaningful computer programs. These include using a computer, writing code, compiling it, running it, testing it as well as debugging. Jenkins describes the results of learning to program as a "sad and depressing state of affairs" based on the outcomes of introductory programming courses. Jenkins notes that learning programming requires two learning styles which must be addressed simultaneously. These two styles consist of the surface learning of syntax rules and the deep learning which must be done for understanding. This combination of learning styles is something most learners are not familiar with and thus presents problems for them. Winslow (1996), in research which compares novices to experts, notes that research has shown that novices are able to understand the grammar of syntax and they are able to write valid syntax but they are unable to to combine these into a valid program. In that same paper Winslow notes that many introductory courses teach 'sophisticated material', yet students are not able to grasp basic concepts like control loops. The progression from a novice to a competent programmer can only occur with the basic understanding of basic concepts along with practice or as Winslow states "the old saw that practice makes perfect has solid psychological basis".

The description of multiple concepts which need to be learned in parallel has been shown to present issues for learners. Sweller (1994) explains this through cognitive load theory. Sweller differentiates the complexity of the material itself (the intrinsic load) and the load imposed by the learning materials themselves (extraneous load). Sweller also notes that any material which does not lead to building of schema (mental models) into long term memory is considered extraneous. Sweller notes that novices become experts by building up schema and eventually bypassing the limited amount of working memory with automation. The same research suggests that the positive effects in minimizing extraneous cognitive load are more likely to be seen in material where there is high element interactivity. The author compares an example of low interactivity (learning a list of words in another language or independent historical facts) to an example of high interactivity (learning how to calculate angles between intersecting lines).

An example of how minor changes in instructional materials can impact learning outcomes is given by what is known as the split attention effect. Sweller

demonstrates this with an example of two geometry diagrams which explain how to calculate angles between intersecting lines. One of the diagrams integrates the text within the diagram while the other separates them (figure 2a,b). The need for the learner to shift their attention from the text to the diagram is a form of extraneous cognitive load (figure 2a). It might appear as a subtle difference, but with the limited amount of working memory, and the complexity of the subject matter, subtle differences can be impactful to the learner.

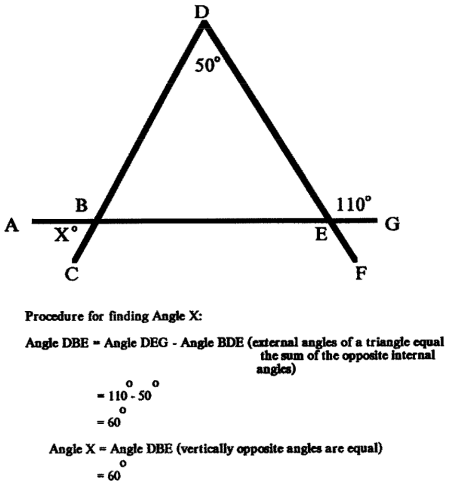

Figure 2a—Conventional Geometry Problem and Solution

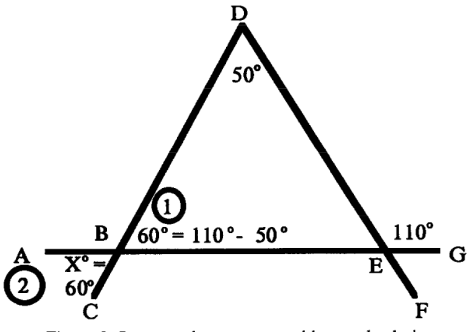

Figure 2b—Integrated Geometry Problem and Solution

3 Impact of Programming Environments on Learning

Based on the concepts of extraneous cognitive load, Mason (2013) has shown that superfluous elements in user interfaces can result in problems for novice learners. These extra elements were considered a distraction in the programming environment. Mason notes that the impact of these negative distractions in the

learners first environment carried over to the next environment which they were introduced to.

The prior discussion assumes an imperfect environment can play a role in distracting the learner from the learning process, but it also assumes the learner actually has an installed environment at their fingertips. Unfortunately, installation issues for novices are problematic (Staubitz et al, 2015). Most web developers use desktop versions of integrated development environments which require both installation, configuration and updates (Aho et al 2011). Issues with the installation of IDE's can be thought of as distractions which novices must spend time on rather than being able to learn programming. Installation of tools becomes both a physical and psychological barrier which can be problematic, especially in a subject area which, as noted by Jenkins (2003), already has a reputation of being difficult. Spending precious time on configuring and installing environments, is time that could be spent doing coding (MacDonald 2013)

4 Traditional Instructional Materials - Application before Comprehension

The lack of novices not having solid mental models before attempting to write code has been noted by Winslow (1996). Bloom's taxonomy, a system for describing an order of proficiency when learning a new topic (Bloom et al, 1956), has been shown to shed further light on the topic (Gomes, 2000, Edwards 2004)

Buck and Stucki (2000) provide examples of skills which might be expected from learners at each level of Bloom's Taxonomy.

- Knowledge - language syntax
- Comprehension - predicting control flow in a program
- Application - writing a unit of code based on a specification
- Analysis - understanding an application with the goal of modification
- Synthesis - designing an application
- Evaluation - system analysis

As noted by Edwards (2004), the writing of code falls under application and synthesis, which becomes a problem when the learner has not mastered comprehension of the subject area. Edwards suggests that by having learners write tests, which will validate the yet to be written code, will aid in comprehension of the topic. By writing tests the learner is forced to hypothesize

about how the program will work which, as noted by Edwards, moves them from 'trial and error to reflection in action'. The ability to reflect on what tests will validate their code will lead to greater comprehension which will be needed to develop the application skills of writing code.

This inversion of Bloom's Taxonomy has also been noted by Gomes et al (2018). This research modified the sequence of instructional tasks so that they conformed to Bloom's Taxonomy. The topic they chose was loops which was the same topic mentioned by Winslow (1996) as being one which is often difficult to grasp by learners or as Winslow phrased it "study after study has shown they don't understand basic loops". A key aspect of instructional sequence was based on having learners read code prior to writing code. This resulted in a better learning experience. The conclusion drawn in this research was that student's need to be able to read programs before they can write them. The students who benefited most from the change were the weaker students.

5 Improving The Tools

5.1 Reducing Environment Barriers

The role of practice is considered critical for novices to progress towards competency (Winslow 1996). It would seem to reason that if we could make it easier for learners to practice, by not having to deal with setup of environments, and if those environments could be designed with minimal extraneous cognitive load, learning outcomes could be improved.

Queriros et al (2021) surveyed the types of tools which are available for learners to practice coding. They make a distinction between tools which need to be installed locally and online environments. In addition, a distinction is made between where the code is actually run. Whether the code runs locally or on a server has both security implications and usability implications. Allowing users to run code on a server can lead to malicious code being run on the server. In addition the delayed response of running the code can result in usability issues as it takes longer to determine if their code is correct. Their analysis is presented in the context of a tool which enables users to practice JavaScript in an online environment.

5.2 Formative Feedback

Summative assessment, a form of feedback, is used in education as a way to grade students. As per Garrison, it can be used to gauge how a student is performing at a particular point in time, but it doesn't provide any corrective action (Garrison 2007). Harlen notes that use of summative assessment in the form of high stakes tests, can have negative motivational effects on learning (Harlen 1997).

As noted earlier, writing tests are recommended as a way for learners to better comprehend material (Edwards, 2004). Writing tests also has the added benefit of providing formative feedback. Formative feedback is feedback which the user can use in the learning process and it is considered quite useful. Formative feedback is adjusted during the learning process. Formative feedback is a form of "practice" which allows the next steps to be determined based on performance (Garrison 2007). Formative feedback can be contrasted with the less useful summative feedback mentioned earlier. It's worth noting that Jenkins (2002) proposes eliminating summative assessment in introductory programming courses.

The reading of code, which also aids in comprehension (Gomez et al, 2018) can also be integrated into the learning process by having learners read not only their code, but the code of their peers. The feedback provided by other learners is a form of peer feedback. Peer feedback has been shown to be a beneficial form of feedback as it comes from learners who are going through the process together as opposed to an authority (Juwah, 2004). Juwah also notes "students who have just learned something are often better able than teachers to explain it to their classmates in a language and in a way that is accessible".

6 Current Web Based Options for Learning and Practice

Online environments are considered to be beneficial for online learners as the frustrations of installation of software presents roadblocks which are difficult to handle in large classes (Staubitz et al, 2015).

Tools which are mentioned in the literature include CodeWars, CodePen, HackerRank, FreeCodeCamp among others (Queiros et al, 2021, Staubitz et al, 2015).

Some of the environments allow developers to write code in an unstructured way. This can be useful for users who already know how to write code. We will focus on the tools which prompt the users with tasks and allow them to complete those tasks while providing some form of feedback.

The tools mentioned previously are not designed for integration into a cohort but are geared more towards users learning on their own. There is a sense of community in the sense of seeing other users' solutions but the community is much broader than a single cohort, which could prove daunting for novices.

Although online environments ease the problem of installation of software, they can still be difficult to navigate. Part of this can be explained by the fact that most of the online environments present the user with many options. These include both language choice and language topic. Assuming both a language and a topic can be chosen, the learner is then faced with additional hurdles in the form of multiple windows for instruction, output, code writing, and testing. In addition the language syntax for testing presents additional information which needs to be processed by the user. There is also additional distraction in the form of tabs and statistics which don't benefit the learning process. It's also worth noting that these commercial tools contain advertisements which can take the user away from the task at hand with a single click. Most of these environments are also configured to run their code on the server which delays the feedback for the user.

A screen shot from one of the more popular environments (https://www.codewars.com/) is shown below:

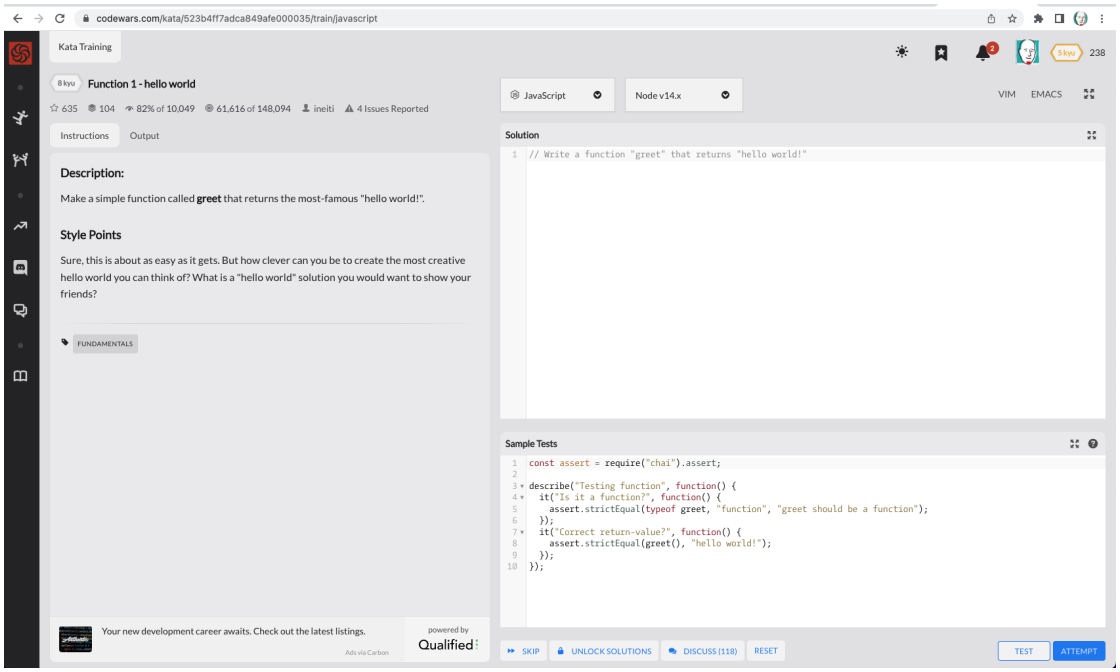

Figure 3- CodeWars

Although the environment saves the user the need for setup, the multiple flyout menus on the left are mostly extraneous to the task at hand. And although tests are often integrated into these environments, as they are here, the language of the testing framework needs to be understood before the tests can be understood.

Not all of the online environments are this complex. Freecodecamp (https://www.freecodecamp.org/) is another environment which was mentioned earlier. A screen shot is shown below.

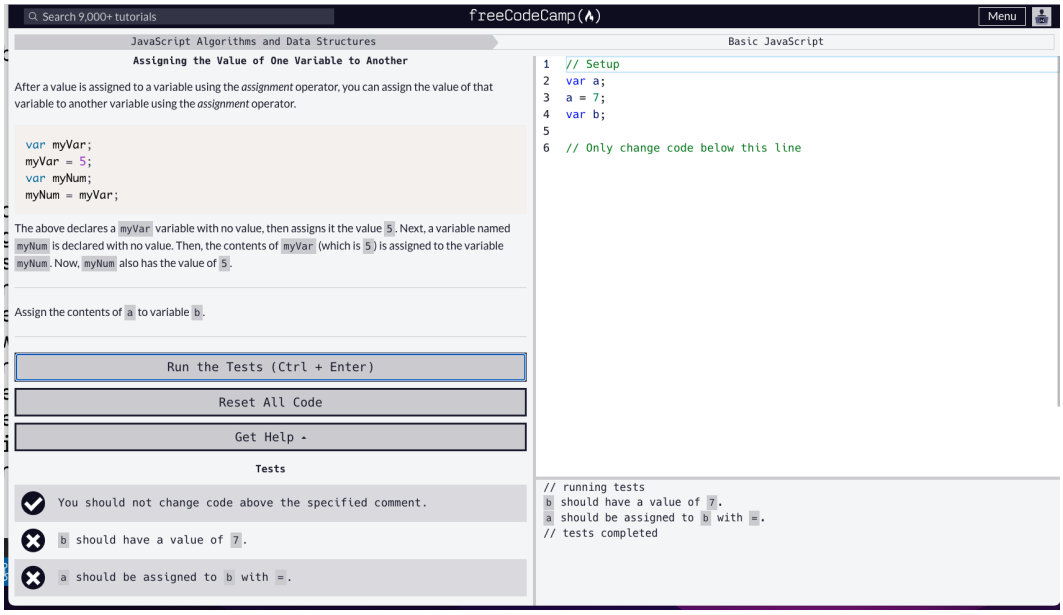

Figure 4 - FreeCodeCamp

This environment is better suited to a novice learner in terms of minimizing the extraneous cognitive load. There are less distractions for the learner. There are also tests which are written in a way that doesn't force the user to learn new syntax which can be considered an improvement over the previous environment. It does however lack the ability for a user to create tests and there is no ability to provide feedback to peers.

In the analysis of online environments, Quierios (2021) proposed a tool (LearnJS) which allowed users to create tests as well as run them. However like the other tools mentioned, this tool is more of an individual practice tool, rather than one which would be used in an online cohort.

7 JS Prompt Solver

JSPrompt Solver (https://www.promptsolver.com/) is an open source project (https://github.com/ericpkatz/js-prompt-solver) which can easily be deployed and configured as a supplemental tool for learning JavaScript. The learner is presented with a series of exercises, based on the current topic being learned in the cohort. These exercises are given in the form of prompts. In the context of JSPrompt Solver these exercises are called code prompts.

7.1 Architecture

The application uses a postgres database, a node server, and a front end user interface built in react. Administrators are able to create courses by uploading topics, coding problems and sample tests in the form. Courses can be assigned to cohorts and students can be assigned to those cohorts. In order to limit the need to administer user credentials, github is leveraged as a third party provider to identify the users. In larger courses this obviates the need for creating user accounts or administering password recovery.

The course is composed of topics and exercises in the form of code prompts are assigned to those topics. Administrators can assign a topic to a cohort which will determine the code prompts they encounter.

The code prompts complexity increases as the user moves through the topics. This enables administrators to remove scaffolding in the form of provided code as the learner progresses.

The code which the user writes is run in their local environment, the JavasScript engine in their web browser. This results in a faster response time which should increase the user experience as mentioned earlier.

7.2 Minimal Distractions

JSPromptSolvers user interface for coding is fairly minimal in comparison to both online editors and the integrated development environments which are sometimes used by novice learners. This enables them to focus on writing the code with minimal distractions. Although scaffolding can be provided there is no auto completion which is sometimes provided to novice learners and can also be considered a form of distraction. The user is able to run their code locally. This falls in line with what was mentioned earlier about security issues which can result in code being run on the server. It also gives the user a faster response which should result in a better user experience. If users like using the system, they are more likely to use it (Queriros et al, 2021).

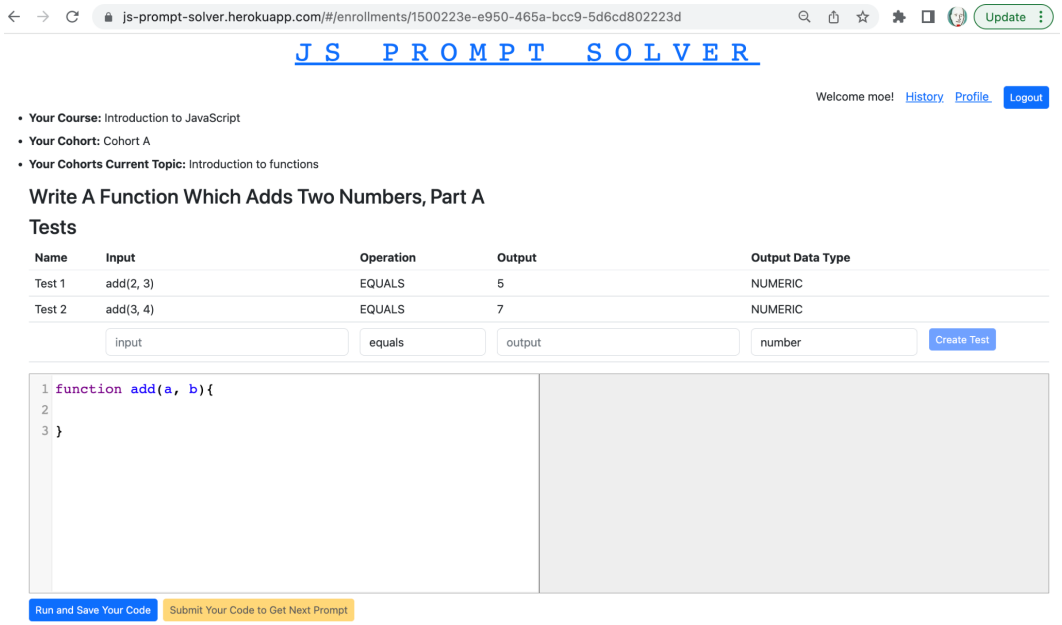

Figure 5 - JS Prompt Solver User Interface

7.3 Writing Tests

As mentioned earlier, the use of tests can not only provide feedback but it can be a way to reflect on the intent of the code. JS Prompt Solver allows the instructional staff to provide tests, but more importantly it allows users to write tests, which have shown to be a way for learners to better comprehend the material. A user is provided with some sample tests, without having to learn any testing framework, but more importantly they are forced to write a test before they move on to the next prompt.

7.4 Peer Feedback

Peer feedback on code prompts can be provided after a user completes a code prompt. Peer feedback is done by reading code written by other users. Reading code has been shown to increase the comprehension of the material. Peer feedback has been shown to be beneficial in the learning process for the receiver of the feedback. In order to promote the use of feedback, learners are reminded to acknowledge the feedback they receive. It also provides a built in scaling mechanism for larger cohorts.

7.5 Practice

Although the learner will practice writing code as they sequentially go through the code prompts which are assigned a specific topic, they can easily see their code history and practice. This allows them to use feedback they might have received from their peers. The role of practice has been mentioned earlier as way of moving novice learners to competent learners (Winslow 1996).

7.5 Administration

Instructional staff can control and reuse content in the form of courses. They can also evaluate how students are doing by examining how far along they are in a particular topic and how much feedback they have given and how many tests they have written. If the learning staff sees that students are struggling on a topic they will be able to adjust accordingly. This becomes adaptive feedback for the instructional staff.

8 Conclusion

Learning computer programming is a difficult topic. The order in which instructional material is presented and the environments in which students write their actual code can play a role in the success or the failure of the learning process. Barriers in the form of complexity in the first coding environment can hamper them as they move forward in the learning process. Designing tools which allow learners to easily write code without distractions and without worrying about installation issues are important aspects of the instructional design process. JSPromptSolver is an example of a tool which will allow students who are part of an online cohort to easily write code with the added benefit of forcing them to write tests to gain better understanding of the material. It will also provide an unobtrusive environment for reading the code of other students in their cohorts and providing them feedback.

9 References

1. Aho, Timo, Adnan Ashraf, Marc Englund, Joni Katajamäki, Johannes Koskinen, Janne Lautamäki, Antti Nieminen, Ivan Porres, and Ilkka Turunen. "Designing IDE as a service." Communications of Cloud Software 1, no. 1 (2011): 1-10.

2. Bloom, B.S., Engelhart, M.D., Furst, E.J., Hill, W.H. & Krathwohl, D.R. (1956). Taxonomy of Educational Objectives: Handbook 1 Cognitive Domain. Longmans, Green and Co Ltd, London.
3. Buck, D., & Stucki, D. J. (2000). Design early considered harmful: graduated exposure to complexity and structure based on levels of cognitive development. ACM SIGCSE Bulletin, 32(1), 75-79.
4. Edwards, S. H. (2004, March). Using software testing to move students from trial-and-error to reflection-in-action. In Proceedings of the 35th SIGCSE technical symposium on Computer science education (pp. 26-30).
5. Garrison, C., & Ehringhaus, M. (2007). Formative and summative assessments in the classroom.
6. Gomes, A., & Correia, F. B. (2018, October). Bloom's taxonomy based approach to learn basic programming loops. In 2018 IEEE Frontiers in Education Conference (FIE) (pp. 1-5). IEEE.
7. Harlen, W., & James, M. (1997). Assessment and learning: differences and relationships between formative and summative assessment. Assessment in education: Principles, policy & practice, 4(3), 365-379.
8. Jenkins, T. (2002, August). On the difficulty of learning to program. In Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences (Vol. 4, No. 2002, pp. 53-58).
9. Juwah , C. , D. Macfarlane-Dick , B. Matthew , D. Nicol , D. Ross , and B. Smith . 2004 . Enhancing student learning through effective formative feedback . The Higher Education Academy Generic Centre, June.
10. MacDonald, B. (2013, November 14). Which Language Should You Learn First? Retrieved from O'Reilly Radar: http://radar.oreilly.com/2013/11/which-language-should-you-learn-first.html
11. Mason, Raina, and Graham Cooper. "Distractions in programming environments." In Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136, pp. 23-30. 2013.
12. Queirós, R., Pinto, M., & Terroso, T. (2021). User Experience Evaluation in a Code Playground (Short Paper). In Second International Computer Programming Education Conference (ICPEC 2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
13. Staubitz, T., Klement, H., Renz, J., Teusner, R., & Meinel, C. (2015, December). Towards practical programming exercises and automated assessment in Massive Open Online Courses. In 2015 IEEE International

Conference on Teaching, Assessment, and Learning for Engineering (TALE) (pp. 23-30). IEEE.
14. Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. Learning and instruction, 4(4), 295-312.
15. Winslow, L. E. (1996). Programming pedagogy—a psychological overview. ACM Sigcse Bulletin, 28(3), 17-22.