Adv Mobile Learn Educ Res, 2026, 6(1): 1759-1766
DOI: 10.25082/AMLER.2026.01.010

RESEARCH ARTICLE

Students' Pre-Instruction Programming Perceptions in Upper-Secondary School: Findings from a Diagnostic Pilot

Sofia Kasotaki

Department of Informatics, School of Sciences, University of Western Macedonia, Kastoria, Greece

Correspondence to: Sofia Kasotaki, Department of Informatics, School of Sciences, University of Western Macedonia, Kastoria, Greece;
Email: kasotaki@gmail.com

Received: November 23, 2025;
Accepted: February 18, 2026;
Published: February 26, 2026.

Citation: Kasotaki, S. (2026). Students' Pre-Instruction Programming Perceptions in Upper-Secondary School: Findings from a Diagnostic Pilot. *Advances in Mobile Learning Educational Research*, 6(1), 1759-1766. https://doi.org/10.25082/AMLER.2026.01.010

Abstract: Pre-instructional diagnostic assessments allows educators to target specific novice misconceptions learners bring to a subject. This data allows them to adjust their initial lesson plans to address these common errors immediately. This pilot study reports on a pre-instruction diagnostic administered to Grade 11 students in one upper-secondary school. The instrument, composed of multiple-choice and True/False items aligned with five conceptual clusters (definition of a program, language recognition, variables and data, basic conditionals, and elementary loop semantics), was designed to reveal common novice difficulties documented in the literature. Analyses of students' responses indicated partial familiarity with simple control constructs but persistent weaknesses in foundational areas, including distinguishing a program from an algorithm, understanding variables as memory locations, and recognizing the role of guard change in loop termination. A consistent format effect favored recognition-based True/False items over multiple-choice discrimination, suggesting that early instruction should bridge from recognition to explanation and short code construction. Although limited by its single-site scope, the pilot provides a practical baseline for refining diagnostic tools and informing initial instructional sequencing in upper-secondary programming.

Keywords: pre-instruction diagnostics, novice misconceptions, introductory programming, upper-secondary education, computing education

1 Introduction

Helping students build a sound conceptual foundation in programming requires understanding how they think about key concepts before formal teaching begins. Novice programmers often rely on fragmented or intuitive reasoning, which may be helpful in some cases but can quickly lead to persistent misconceptions if not addressed early (Luxton-Reilly et al., 2018; Sorva, 2013). Prior research shows that beginners frequently construct incomplete or unstable mental models of how programs work, particularly in relation to state changes, control structures, and the underlying semantics of execution. Once these misconceptions become entrenched, they can interfere with later learning and are often difficult to correct.

A pre-instruction diagnostic offers an opportunity to surface these underlying conceptions. Teachers should use these diagnostic results to identify specific gaps in student understanding, allowing them to adjust their instructional sequencing accordingly. When diagnostic items are explicitly mapped to conceptual targets, such as variables, conditionals, or loops, the collected responses provide insight into learners' thinking and inform decisions about instructional sequencing, representational choices, and pacing (Kecskemety et al., 2021; Parker et al., 2023).

In Greece, these considerations are especially relevant. Informatics and programming have been formally integrated into the upper-secondary curriculum, with clear expectations that students develop computational thinking competencies and basic programming literacy by the end of their schooling (Eurydice, 2025). As schools adjust to this increased emphasis, understanding students' starting points becomes essential. This study presents the findings from a short diagnostic administered to Grade 11 students before any instruction in programming during the academic year. The goal is to document their conceptual baselines and provide evidence that can guide instructional design in early lessons.

2 Theoretical Framework

2.1 Novice Misconceptions in Programming

(1) Variables, Assignment, and Program State: One of the most well-documented novice challenges concerns the interpretation of variables. Since beginners often mistake a variable for a static label, instructors should show how variables actually map to dynamic memory locations during the first few lessons. This misconception undermines correct reasoning about assignment, state changes, and the flow of computation (Sorva, 2013). A related difficulty is the distinction between assignment and equality. For example, expressions like '$x = x + 1$' often confuse beginners because they read the '=' symbol as a sign of mathematical equality. (Qian & Lehman, 2017; Brown & Altadmri, 2014). To address this, teachers should explain students that this is actually a command to update a value in memory, not a balanced equation.

(2) Control Flow and Branching: Misinterpretations of conditional statements further complicate novice reasoning. Students may believe that the else part of an if/else structure executes alongside or immediately after the if block, or they may attribute side effects to branches that are unrelated to condition evaluation (Luxton-Reilly et al., 2018). Additionally, many rely on natural-language interpretations of conditions rather than formal Boolean semantics, which becomes especially problematic for compound expressions.

(3) Loops and Guard Conditions: Loop semantics are another set of common difficulties. Students frequently generalize from one loop type to another. For example, assuming that a for loop and a while loop behave identically or that both imply a fixed number of iterations (Qian & Lehman, 2017). Understanding of loop termination is also often incomplete. Many beginners fail to consider whether the loop guard condition will change over time, making it difficult to recognize or avoid infinite loops.

(4) Operator Precedence, Boolean Logic, and Tracing: Difficulties increase when arithmetic, comparison, and logical operators combine–precisely the context requiring accurate tracing. Large-scale datasets highlight errors related to precedence, mismatched tokens, and evaluation-order assumptions; misconceptions about equality and Boolean logic often persist without targeted instruction (Altadmri & Brown, 2015; Brown & Altadmri, 2014; Wilson, 2016).

These recurring patterns support the design of diagnostics where distractors embody specific misconceptions, allowing student responses to reveal their underlying mental models, an approach now common in computing concept inventories (Kecskemety et al., 2021; Parker et al., 2023). Additional research in secondary education further shows that novice learners benefit from scaffolded, visual, and block-based programming environments (Kalogiannakis & Papadakis, 2019). Studies with secondary novices also show that combining tangible robotics with beginner-friendly app development can lower entry barriers and support conceptual engagement (Papadakis & Orfanakis, 2016). Comparative classroom evidence further indicates that novice-oriented environments such as App Inventor and Alice can differentially support early conceptual understanding in secondary settings (Papadakis & Orfanakis, 2018).

2.2 Role of Pre-Instruction Diagnostics

Pre-instruction diagnostics are not intended to measure proficiency but rather to reveal novice interpretations of key programming concepts. They serve as windows into students' mental models at the earliest stage of learning, before formal instruction shapes their understanding (Qian & Lehman, 2017; Sorva, 2013). Misconceptions in programming often depend on students' prior experiences, such as earlier schooling, everyday language, and classroom contexts. Because of this, diagnostics that focus on specific concepts like variables, control flow, or loops are far more helpful than trying to measure a general 'aptitude.' This idea is supported by major reviews on novice programming difficulties (Qian & Lehman, 2017; Luxton-Reilly et al., 2018) and by recent work on how concept-focused assessments are developed and validated to identify particular misconceptions in computing education (Ali et al., 2023).

When instructors know which conceptual barriers are likely to impede progress, they can adapt lesson plans and introduce representational scaffolds, such as state tables, diagrams, or simplified models of execution, to support conceptual clarity (Qian & Lehman, 2019). In addition, collaborative structures like pair programming have been shown to benefit secondary-level novices' engagement and understanding (Papadakis, 2018b). Research on instructional design also shows that structured collaborative practices, such as pair programming, can improve novice programmers' understanding and engagement (Papadakis, 2018b). In this regard, pre-instruction diagnostics function as practical instruments for linking curriculum design to learners' starting points.

2.3 Greek Context

In recent years, Greece has expanded its emphasis on Informatics across the secondary curriculum. Python and algorithmic thinking feature prominently in official documents and teaching materials reflecting broader European policy directions toward digital literacy (Eurydice, 2025). The Greek

Lyceum curriculum (FEK B 5932/16122021) emphasizes structured exposure to core computing concepts, modern pedagogies, and digital competencies for all (Eurydice, 2025). In the Greek upper-secondary context, recent system-level reviews point to opportunities to deepen conceptual engagement and strengthen the alignment of digital resources, pedagogy, and assessment — reinforcing the case for diagnostic evidence to inform teaching. Comparative and country-level analyses (Eurydice, 2025; European School-net, 2021) highlight the ongoing modernization of Informatics in Lyceum alongside needs around curriculum implementation, teacher support, and learning materials. Furthermore, textbook analyses indicate the presence of gendered representations in Greek computer science materials, suggesting additional areas where curriculum design can be improved (Papadakis, 2018a).

National and European policy reports emphasize using diagnostic evidence to personalize instruction and support informed pedagogical decision-making (European School-net, 2021). A pre-instruction diagnostic administered to Grade 11 students is therefore well aligned with national goals and offers actionable insights for teachers preparing to introduce programming to diverse classroom contexts.

3 Diagnostic Focus and Instrument Design

The diagnostic used in this study was deliberately designed as a conceptual probe, not as a performance-based test. Its purpose was to capture students' intuitive understandings and possible misconceptions within five conceptual areas:

(1) Definition and purpose of a program,
(2) Recognition of programming languages,
(3) Variables and data types,
(4) Basic control-flow semantics,
(5) Elementary loop semantics.

Guided by prior research on novice programming misconceptions and aligned with the learning objectives of upper-secondary introductory programming curricula, this study's diagnostic instrument was designed to probe foundational perceptions rather than advanced procedural skills. The items targeted five core areas that commonly present conceptual challenges for beginners.

First, the diagnostic examined students' basic understanding of what a program is, including its purpose and functional nature. Second, it assessed introductory knowledge of programming languages, focusing on students' ability to recognize commonly used languages such as Python. Third, the instrument explored fundamental concepts of variables and data, including what a variable represents and whether it can store different types of values. Fourth, it addressed basic control-flow semantics, such as the meaning of if and else, the role of conditional statements, and the interpretation of simple Boolean conditions (*e.g.*, if $(a > b)$). Fifth, the questionnaire targeted elementary loop semantics, including the function of the while and for loops and conditions that can result in non-terminating iteration (*e.g.*, a guard that never changes).

These conceptual clusters correspond directly to the structure of the administered items. They are intended to surface introductory-level reasoning and misconceptions, rather than to assess advanced abilities such as code tracing or operator-precedence logic. Although the diagnostic is intentionally simpler than established concept inventories in computing education, its design reflects a widely supported methodological principle: well-targeted items can reveal stable novice misconceptions in areas such as variables, conditionals, and loops, providing insight into learners' mental models prior to instruction (Kecskemety et al., 2021; Parker et al., 2023).

4 Methods

4.1 Participants and Context

The pilot was conducted during the 2023-2024 school year in a single Greek upper-secondary school. Participants were Grade 11 students enrolled in the standard Informatics course. Participation was voluntary and anonymous under official authorization for a pilot research activity. Challenges related to participation and response reliability were well-documented in educational survey research (Lavidas et al., 2022), although in this study all students in the targeted class completed the diagnostic.

4.2 Instrument

Students completed a pre-instruction diagnostic questionnaire composed solely of multiple-choice and True/False items addressing the five conceptual clusters. The instrument is intentionally introductory and targets perceptions and conceptual reasoning, not programming fluency.

4.3 Procedure

The diagnostic was administered during a scheduled class session before the start of programming instruction. Students completed the instrument individually and without preparation. Data were exported for analysis.

4.4 Data Analysis

Analyses were quantitative. Descriptive statistics were calculated for each item, including accuracy rates and distractor frequencies. When relevant, results were summarized by gender for descriptive comparisons, which is appropriate to the pilot scope. Item-level distributions were inspected to identify recurring misconceptions (*e.g.*, program *vs.* algorithm; fixed-count *vs.* condition-controlled iteration).

Given the structure of the instrument, the analysis was entirely quantitative. Descriptive statistics were calculated for each item, including the proportion of correct and incorrect responses, overall performance scores, and identifiable patterns in students' selections of distractors. Where relevant, results were also examined by student gender to produce simple group summaries; these comparisons were treated as descriptive due to the pilot scope of the study.

In addition to overall accuracy measures, item-level response distributions were inspected to reveal recurring misconceptions or areas of conceptual difficulty. These analyses were used to build an initial profile of students' pre-instruction understanding and to identify potential focal points for subsequent teaching. No qualitative coding or thematic analysis were undertaken, as the study did not include interviews or open-ended responses.

5 Results

5.1 Overall Performance

Before instruction, students' overall performance on the diagnostic was low to moderate. Across the 72 participants, total scores ranged from 1 to 11 correct responses out of 13, indicating substantial variability in pre-instruction preparedness. The mean total score was 6.11 (SD = 2.23), corresponding to an average accuracy of approximately 47%, while the median score was 6. These results show that many students entered the introductory programming course with only a partial understanding of the basic ideas involved. This pattern aligns with previous studies reporting that learners often start introductory programming with incomplete or unstable mental models (Sorva, 2013; Luxton-Reilly et al., 2018). The variation in scores highlights the mixed levels of conceptual readiness in the group, underscoring the importance of using a diagnostic assessment before instruction begins.

5.2 Item-Level Performance

Item-level analyses revealed substantial variation in students' conceptual understanding across the diagnostic. Several items assessing foundational definitions and state-related reasoning produced notably low accuracy rates, while items requiring recognition of simple control-flow behavior yielded somewhat higher performance. Item-level percentages and the most frequent distractors are summarized in Table 1.

Table 1 Item-Level Accuracy and Most Frequent Distractors

Item (Short Label) *Most Frequent Distractor (%)*	% Correct
What is a program? *"Rules to solve a problem" (40.3%)*	27.8%
Which is a programming language? *Windows (21%), Android (10%), Google (9%)*	44.4%
What is a variable? *"Command executed multiple times" (36.1%)*	22.2%
What does else statement do? *"Executes when true" / "Creates a loop" (~24%)*	40.8%
Function of while *"Predetermined iterations" (33.3%)*	37.5%
Infinite loop cause *"Use of if statement" (20.8%)*	54.2%
What happens in if ($a > b$)? *"Does nothing" (19.7%)*	62.5%
Variable stores multiple data types (T/F)	59.7%
If checks if condition is true/false (T/F)	58.3%
For executes until condition false (T/F)	45.1%
While executes while true (T/F)	65.3%
Infinite loop if condition never changes (T/F)	62.5%
If–else can contain nested if statements (T/F)	56.9%

The weakest performance was observed on items targeting program definition and variables. Only 27.8% of students correctly identified a program as "a set of instructions executed by a computer", whereas the most frequent distractor (40.3%) defined a program as "a set of rules followed to solve a problem", indicating a strong confusion between programs and algorithms, a misconception well-documented in the literature on novice cognition (Sorva, 2013). Similarly, for the question "What is a variable?", only 22.2% selected the correct interpretation, while the dominant distractor (36.1%) framed a variable as "a command that runs multiple times". This pattern reflects a deep conceptual substitution in which students map the unfamiliar idea of state to more familiar notions such as repetition or control.

Understanding of conditional structures was also uneven. Although students could often recognize the truth-checking purpose of an if statement (58.3% on the True/False item), only 40.8% correctly identified the purpose of else. Many selected distractors indicating that else executes when the condition is true (23.9%) or that it "creates a loop" (23.9%), illustrating confusion between branching and iteration, another common novice pattern (Luxton-Reilly et al., 2018).

Performance on loop semantics showed similar inconsistencies. While 65.3% recognized that a while loop executes while its condition is true, only 37.5% selected the correct multiple-choice definition. A substantial proportion (33.3%) instead indicated that a while loop performs a predetermined number of iterations, again conflating while with for. This aligns with research showing that learners frequently over-generalize across loop constructs (Qian & Lehman, 2017). Students showed better understanding of infinite loops (54.2% correct), though nearly one in five believed that the use of an if statement can cause non-termination, pointing to a lack of clarity about the role of guard change.

For the item interpreting the statement if $(a > b)$, 62.5% selected the correct answer. Notably, 19.7% believed the statement "executes nothing," suggesting incomplete understanding of how conditionals govern control flow, especially in abstract pseudo-code-like forms.

Overall, item-level patterns paint a consistent picture: students enter programming instruction with partial recognition of isolated control-flow behaviors, but with significant weaknesses in definitional concepts, the nature of state, and the semantics of loop guards. These misconceptions are not random errors, they align closely with documented novice reasoning tendencies and underscore the conceptual starting points that instruction must address.

5.3 Misconception Patterns

Patterns of misconceptions became evident when examining the distractors chosen by students across the diagnostic (see Table 1). These distractors are not random mistakes but reveal stable novice mental models that align closely with well-documented difficulties in introductory programming.

5.3.1 Program *vs*. Algorithm (and Program *vs*. OS/Application)

In the "What is a program?" item, the most common distractor was "a set of rules followed to solve a problem" (40.3%), while only 27.8% selected the correct executable definition. This reflects a strong confusion between programs and algorithms, in which students interpret "program" through more familiar everyday notions of a "procedure" or "rule sequence." A substantial minority additionally confused a program with an operating system or tools, indicating difficulty distinguishing between software categories. Such patterns are frequently reported in research on novice mental models (Sorva, 2013; Luxton-Reilly et al., 2018).

5.3.2 Variable as Action, not State

The item on variables revealed one of the clearest misconceptions: 36.1% selected "a command executed multiple times," and other distractors linked variables to input or condition checking. This shows that many students conceptualize a variable as an action rather than a memory location that holds a value. This difficulty (an inability to mentally represent program state and follow state changes) is a central issue for beginners (Qian & Lehman, 2017; Sorva, 2013).

5.3.3 Branching Confused with Looping (Else Semantics)

Only 40.8% correctly identified the purpose of the else branch. Nearly equal proportions believed that else "executes when the condition is true" or "creates a loop" (~24% each). This indicates two misconception patterns:

(1) Logical reversal (misinterpreting true/false conditions);

(2) Structural confusion between branching and repetition, a well-known novice tendency (Luxton-Reilly et al., 2018).

5.3.4 Loop Semantics and the Missing Notion of Guard Change

The multiple-choice item on the function of a while loop strongly revealed over-generalization: 33.3% believed a while loop performs a fixed number of iterations, essentially describing a for loop.

This suggests that students do not yet differentiate the idea of a condition-controlled loop.

Although most students correctly recognized that an infinite loop may occur when a guard never changes (54.2%), 20.8% attributed non-termination to the use of if, indicating that the role of state update within the loop is not yet understood.

5.3.5 Partial Recognition without Understanding the Execution Mechanism

For the item interpreting if $(a > b)$, 62.5% answered correctly, but 19.7% believed the statement "does nothing." This illustrates partial recognition: students may recall what a conditional does in general terms but fail to map a specific condition to its execution behavior in an abstract, pseudo-code context. Such gaps highlight the difference between recognition of an idea and understanding the mechanism behind it.

5.4 Format Effect (MC *vs* T/F)

A clear format effect emerged across the diagnostic. Students performed substantially better on True/False items than on multiple-choice items that required distinguishing between several closely related alternatives. Aggregated accuracy was $\approx$ 59.5% for True/False compared with $\approx$ 41.3% for multiple-choice. This pattern was also visible within topical areas. For example, students more readily endorsed the statement that a while loop executes while its condition is true (T/F) than selected the correct multiple-choice definition of while; likewise, they more often recognized that a non-changing guard can cause an infinite loop (T/F) than chose the correct cause among multiple options (MC). (see Table 2)

Table 2 Accuracy by Item Format

Item format	Mean % correct
Multiplechoice	~41.3%
True/False	~59.5%

Note: Values are aggregated over all items of that format in the diagnostic.

Pedagogically, this format effect suggests that many students can recognize a correct proposition in isolation but struggle when they must discriminate between near-miss alternatives — a hallmark of fragile conceptual knowledge. Early instruction can build on this by moving step-by-step from recognition to explanation and then to construction. Assessment should move from binary recognition to identify initial misconceptions, followed by a brief written explanation of the student's reasoning. Finally, students trace a short code snippet or write a small example to demonstrate they can apply the concept in practice. This gradual progression helps strengthen understanding and makes students less likely to fall prey for common misconceptions.

5.5 Gender Differences (Descriptive)

The sample was balanced by gender (36 males, 36 females). At the whole-test level, males averaged 52.14% (median 53.85%) and females 47.22% (median 50.00%), indicating a small standardized mean difference (Hedges' g $\approx$ 0.26) that indicates a modest male advantage at baseline. A similar descriptive pattern appeared when performance was summarized by item format: males outperformed females on multiple-choice items (43.3% *vs.* 39.3%) and on True/False items (62.5% *vs.* 56.5%). At the item level, the largest gaps favored males on programming-language recognition (52.8% *vs.* 36.1%), the statement that "if checks whether a condition is true or false" (66.7% *vs.* 50.0%), and the interpretation of if $(a > b)$ (69.4% *vs.* 55.6%). A small female advantage emerged on the variable definition item (25.0% *vs.* 19.4%), although performance on that constructs was low for both groups. These comparisons are descriptive only and should be interpreted with caution given the single-site pilot and the introductory scope of the instrument; they serve simply to give context about baseline differences, not to make statistical claims.

6 Discussion

This diagnostic pilot provides a clear picture of how upper-secondary students approach programming before receiving formal instruction. The results show that learners enter the classroom with some basic intuitions, particularly about simple control flow, yet lack stable understanding of core conceptual structures such as program purpose, variable state, and the conditions that govern loop execution. These difficulties align with well-established findings in introductory programming research, where novices often rely on informal or everyday reasoning instead of the underlying computational mechanisms that determine program behavior (Luxton-Reilly et al., 2018; Sorva, 2013).

A notable pattern across items was students' tendency to replace unfamiliar abstractions with more accessible, but incorrect, everyday analogies. Confusions such as treating a program as an algorithm, interpreting a variable as an action, or assuming that while loops behave like fixed-count for loops illustrate how learners try to make sense of new concepts using the reasoning tools already available to them. Identifying these patterns early is important, as instruction that is built on mistaken assumptions about students' prior knowledge may inadvertently reinforce misconceptions.

The format effect observed in this study further highlights the fragility of students' conceptual understanding. Higher accuracy on True/False items suggests that many students can recognize correct statements in isolation but struggle to distinguish correct reasoning from closely related alternatives in multiple-choice formats. Early programming instruction may therefore benefit from a gradual sequence that moves from recognition to explanation and then to short tracing or simple coding tasks. Such an approach encourages learners to actively work with program behavior, supporting deeper and more durable understanding.

Gender differences appeared in several descriptive comparisons but must be interpreted carefully. Given the single-site scope of the pilot and its diagnostic purpose, the study does not support inferential claims about gender. The descriptive summaries are included solely to illustrate the range of starting points within the class and to emphasize the importance of understanding individual learners' conceptual baselines.

Overall, the findings highlight the value of pre-instruction diagnostics as tools for uncovering students' initial conceptual models. By making misconceptions visible before instruction begins, teachers can more effectively plan early lessons, and address the most essential concepts first, taking into account students' actual prior knowledge rather than relying on assumptions.

7 Conclusion

This pilot study shows that a short, focused diagnostic can provide teachers with a clear understanding of what students understand before they start learning programming. The results reveal that many upper-secondary students begin with some basic ideas about programming but still have important gaps in areas like what a program is, how variables work, and how loops depend on changing conditions. Knowing this from the start can help teachers plan lessons that address these misunderstandings early.

Based on these findings, early instruction should focus on building clear, solid foundations. Teachers may need to explain the difference between a program and an algorithm, show how variables store and change values, and highlight how different kinds of loops work. A gradual approach that begins with simple recognition tasks, moves on to asking students to explain their thinking, and ends with having them try small traces or code examples can help students develop a stronger and more accurate understanding.

Although this was a small pilot in one school, it shows that using a diagnostic at the beginning of a programming unit is both practical and useful. Future studies could try the same approach in more schools, include additional concepts like tracing and operator precedence, and explore how early activities influence learning over time. Administering targeted diagnostics at the start of a course allows for precise adjustments to instruction. By grounding the curriculum in this data, teachers can prevent early misconceptions from turning into persistent errors in later programming modules.

8 Limitations

This study was a small-scale pilot conducted in a single upper-secondary school, so the findings are preliminary and cannot be generalized to all Greek schools or broader student populations. The diagnostic focused only on introductory programming concepts, such as program purpose, variables, simple conditionals, and basic loop semantics, and did not assess more advanced reasoning like tracing or operator precedence. As a result, the data reflected students' initial perceptions rather than their broader conceptual understanding.

The study relied solely on students' item choices without qualitative evidence to explain their reasoning, which limits the depth of interpretation. Because the diagnostic was administered only before instruction, no conclusions could be drawn about learning progress. More broadly, performance-based studies must acknowledge interpretive and contextual constraints, as noted in the research-ethics literature (Petousi & Sifaki, 2020). Despite these limitations, the pilot offered a useful baseline for refining the instrument and informing future, larger-scale work.

Conflicts of Interest

The author declares no conflict of interest.

References

Ali, M., Ghosh, S., Rao, P., Dhegaskar, R., Jawort, S., Medler, A., Shi, M., & Dasgupta, S. (2023). Taking Stock of Concept Inventories in Computing Education: A Systematic Literature Review. Proceedings of the 2023 ACM Conference on International Computing Education Research V.1, 397–415. https://doi.org/10.1145/3568813.3600120

Altadmri, A., & Brown, N. C. C. (2015). 37 Million Compilations. Proceedings of the 46th ACM Technical Symposium on Computer Science Education, 522–527. https://doi.org/10.1145/2676723.2677258

Brown, N. C. C., & Altadmri, A. (2014). Investigating novice programming mistakes. Proceedings of the Tenth Annual Conference on International Computing Education Research, 43–50. https://doi.org/10.1145/2632320.2632343

European Schoolnet. (2021). Supporting digital education in Greece: Towards a digitally enhanced educational ecosystem. European Schoolnet.

Eurydice. (2025). Teaching and learning in general upper secondary education (Greece). Education, Audiovisual and Culture Executive Agency.

Kalogiannakis, M., & Papadakis, S. (2019). Pre-service kindergarten teachers' acceptance of "ScratchJr" as a tool for learning and teaching computational thinking and science education. The Journal of Emergent Science, 15, 31–34.

Kecskemety, K., Barach, A., Jenkins, C., & Gunawardena, S. (n.d.). Using Programming Concept Inventory Assessments: Findings in a First-year Engineering Course. 2021 ASEE Virtual Annual Conference Content Access Proceedings. https://doi.org/10.18260/1-2-37995

Lavidas, K., Petropoulou, A., Papadakis, S., Apostolou, Z., Komis, V., Jimoyiannis, A., & Gialamas, V. (2022). Factors Affecting Response Rates of the Web Survey with Teachers. Computers, 11(9), 127. https://doi.org/10.3390/computers11090127

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., & Szabo, C. (2018). Introductory programming: a systematic literature review. Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, 55–106. https://doi.org/10.1145/3293881.3295779

Papadakis, S. (2018a). Gender stereotypes in Greek computer science school textbooks. International Journal of Teaching and Case Studies, 9(1), 48. https://doi.org/10.1504/ijtcs.2018.090196

Papadakis, S. (2018b). Is Pair Programming More Effective than Solo Programming for Secondary Education Novice Programmers? International Journal of Web-Based Learning and Teaching Technologies, 13(1), 1–16. https://doi.org/10.4018/ijwltt.2018010101

Papadakis, S., & Orfanakis, V. (2017). The Combined Use of Lego Mindstorms NXT and App Inventor for Teaching Novice Programmers. Educational Robotics in the Makers Era, 193–204. https://doi.org/10.1007/978-3-319-55553-9_15

Papadakis, S., & Orfanakis, V. (2018). Comparing novice programing environments for use in secondary education: App Inventor for Android vs. Alice. International Journal of Technology Enhanced Learning, 10(1/2), 44. https://doi.org/10.1504/ijtel.2018.088333

Parker, M. C., Davidson, M. J., Kao, Y. S., Margulieux, L. E., Tidler, Z. R., & Vahrenhold, J. (2023). Toward CS1 Content Subscales: A Mixed-Methods Analysis of an Introductory Computing Assessment. Proceedings of the 23rd Koli Calling International Conference on Computing Education Research, 1–13. https://doi.org/10.1145/3631802.3631828

Petousi, V., & Sifaki, E. (2020). Contextualising harm in the framework of research misconduct. Findings from discourse analysis of scientific publications. International Journal of Sustainable Development, 23(3/4), 149. https://doi.org/10.1504/ijsd.2020.115206

Qian, Y., & Lehman, J. (2017). Students' Misconceptions and Other Difficulties in Introductory Programming. ACM Transactions on Computing Education, 18(1), 1–24. https://doi.org/10.1145/3077618

Qian, Y., & Lehman, J. D. (2019). Using Targeted Feedback to Address Common Student Misconceptions in Introductory Programming: A Data-Driven Approach. Sage Open, 9(4). https://doi.org/10.1177/2158244019885136

Sorva, J. (2013). Notional machines and introductory programming education. ACM Transactions on Computing Education, 13(2), 1–31. https://doi.org/10.1145/2483710.2483713

Wilson, G. (2016, June 9). Novice programming mistakes. It Will Never Work in Theory. https://neverworkintheory.org