Vilnius University Press

Contents lists available at Vilnius University Press

Acta Paedagogica Vilnensia
2026, vol. 56, pp. 166–183

ISSN 1392-5016 eISSN 1648-665X
DOI: https://doi.org/10.15388/ActPaed.2026.56.10

Self-Reported Programming Learning Difficulties Among Pre-Service Mathematics Teachers: Evidence from an Algerian Teacher-Education Context

Sara Chellali

Department of Computer Science, Ecole Normale Supérieure, Laghouat, Algeria
Applied Chemical and Physical Sciences Laboratory,
Ecole Normale Supérieure, 03000 Laghouat, Algeria
s.chellali@ens-lagh.dz
https://orcid.org/0000-0002-4533-2741
https://ror.org/004n81v75

Abstract. This study examines self-reported programming difficulties among 247 pre-service mathematics teachers in Algeria. By using a context-adapted Arabic questionnaire, the paper found that difficulties were better described as conceptual and application/debugging-related patterns. Debugging, theory-to-practice transfer, and coding frustration were the main challenges. The findings point to guided practice, explicit debugging support, mathematics-linked tasks, and differentiated support. The study contributes to teacher education and programming education research by highlighting how programming difficulties in non-specialist mathematics teacher education are shaped more by application and debugging challenges than by conceptual understanding alone.
Keywords: Programming education, mathematics teacher education, learning difficulties, motivational orientation, instructional design, Algeria.

Būsimų matematikos mokytojų patiriami programavimo mokymosi sunkumai: Alžyro mokytojų rengimo programos duomenys

Santrauka. Šiame tyrime analizuojami 247 būsimų matematikos mokytojų Alžyre patiriami programavimo sunkumai. Naudojant kontekstui modifikuotą klausimyną arabų kalba nustatyta, kad tyrimo dalyviams kylantys sunkumai daugiausia susiję su teoriniais principais, jų taikymu ir klaidų šalinimu. Pagrindiniai iššūkiai buvo klaidų taisymas, teorijos perkėlimas į praktiką ir nusivylimas kodavimu. Rezultatai rodo, kad svarbiausia dalyviams yra vadovaujama praktika, aiški pagalba taisant klaidas, su matematika susietos užduotys ir diferencijuota pagalba. Šie radiniai prisideda prie mokytojų rengimo ir programavimo mokymų tyrimų ir atskleidžia tai, kad programavimo sunkumai nespecialistų matematikos mokytojų rengimo programoje labiau susiję su taikymo ir klaidų taisymo sunkumais, o ne vien tik su teorinių principų supratimu.
Pagrindiniai žodžiai: programavimo mokymai, matematikos mokytojų rengimas, mokymosi sunkumai, motyvacinė orientacija, mokymo projektavimas, Alžyras

Received: 30/04/2026. **Accepted:** 04/06/2026

1. Introduction

Programming currently plays a broader educational role beyond specialist computing, serving as a medium for modelling, representing procedures, exploring patterns, and supporting problem solving. This is especially relevant in mathematics teacher education, where programming can support algorithms, numerical methods, simulation, and dynamic representation. Difficulty in learning programming in this context is therefore not only a technical issue, but also an instructional design issue affecting how future teachers engage with programming as part of their disciplinary preparation.

At the same time, learning to program remains difficult for many beginners. Prior research has shown that novice learners often struggle with basic structures, tracing, translating ideas into algorithms, debugging, and persisting when the code does not work as expected (Robins et al., 2003; Derus & Zamzuri, 2012; Luxton-Reilly et al., 2018). These challenges are shaped not only by learners' knowledge, but also by how programming activities are sequenced, scaffolded, practised, and supported.

The issue becomes more acute in non-specialist settings. When students' primary academic identity lies outside computer science, programming is often positioned within a broader professional curriculum, with limited exposure, constrained practice time, and competing disciplinary priorities. In such contexts, learners may understand examples during explanation while still struggling to apply concepts independently, interpret errors, or debug incomplete solutions (Jenkins, 2002; Putri et al., 2024; Ngadengon et al., 2025). For teacher education and programming education research, the key question is therefore not only whether students report difficulty, but what kind of difficulty they report, and what this pattern suggests for the design of programming learning environments.

A design-oriented perspective is useful here. If the main bottleneck lies in conceptual access to basic structures, instructional responses may need to prioritize sequencing, explanation, and representational clarity. If the larger bottleneck lies in the theory-to-practice transfer, debugging, and coping with coding breakdowns, different priorities follow: guided application, continuous practice, explicit debugging pedagogy, and tasks that connect programming to meaningful disciplinary purposes (Lister et al., 2004; Krpan et al., 2015; Ou et al., 2023). This distinction matters because different self-reported difficulty patterns imply different educational responses.

Within Algeria and in the wider Arab/North African context, programming learning in teacher education remains underrepresented in the international literature. Existing work has more often addressed educational technology, institutional reform, or school-level initiatives than the ways in which future teachers experience programming as learners. Against this background, the present study examines one Algerian Higher School for Teachers as a context-bound case of non-specialist programming education. It asks whether reported difficulty is better understood as one broad tendency or as a distinction between conceptual difficulty and application/debugging-related difficulty, and it considers motivation, support indicators, and student-reported instructional preferences. The aim is not to generalize prevalence, but to clarify difficulty patterns that may inform the instructional design.

The study was guided by the following research questions:

- RQ1. What difficulty patterns are reported at the item and subscale levels, and does exploratory factor analysis distinguish conceptual difficulty from application/debugging-related difficulty?
- RQ2. What are the levels of motivation and support indicators reported in this context, and what preferred learning methods and suggested instructional improvements do students identify?
- RQ3. Do conceptual difficulty, application/debugging-related difficulty, and motivation differ according to gender, second-year versus third-year programme context, and prior programming experience?
- RQ4. Which student characteristics are associated with conceptual difficulty, application/debugging-related difficulty, and motivation after simultaneous adjustment?

2. Literature Background and Instructional Design Framing

Programming learning is widely recognized as a multidimensional educational challenge. Research on introductory programming has shown that learners struggle not only with the syntax and basic control structures, but also with tracing program behaviour, translating problems into algorithmic steps, debugging errors, and applying abstract concepts in executable form (Derus & Zamzuri, 2012; Luxton-Reilly et al., 2018; Robins et al., 2003). Difficulty in programming therefore spans conceptual understanding, procedural execution, and learners' responses to repeated breakdowns during problem solving.

This complexity is especially important in non-specialist settings. For learners whose primary academic identity is not computing, programming is often encountered within a broader professional curriculum rather than a dedicated computer science pathway. Under conditions of limited exposure, constrained practice time, and competing disciplinary priorities, the fragile point is often the transition from explanation to implementation and from correct examples to independent problem solving (Jenkins, 2002; Ngadengon et al., 2025; Putri et al., 2024).

Learning to program involves more than acquiring knowledge of syntax or language-specific commands. Research on novice programmers suggests that programming requires learners to construct mental models of program behaviour, decompose problems into manageable steps, and translate abstract reasoning into executable procedures. Difficulties often emerge not because students fail to memorize concepts, but because they struggle to coordinate conceptual understanding, algorithmic thinking, and procedural implementation. As a result, programming learning is increasingly viewed as a process of knowledge construction and problem solving rather than the simple acquisition of technical skills (Robins et al., 2003; Luxton-Reilly et al., 2018).

From this perspective, successful programming learning depends on the learners' ability to move between understanding and action. Students may understand a concept when it is explained or demonstrated, yet still experience difficulty when required to

apply it independently, diagnose errors, or revise unsuccessful solutions. Debugging, tracing, and iterative problem solving are therefore not peripheral activities but central components of programming competence. These processes require persistence, logical reasoning, and the ability to reflect on the relationship between intended and actual program behaviour.

This perspective is particularly relevant in teacher education. Pre-service mathematics teachers typically engage with programming within a broader professional curriculum whose primary focus is not computer science. Consequently, programming learning occurs alongside disciplinary, pedagogical, and professional demands. In such contexts, the educational value of programming extends beyond technical competence, as programming can support mathematical reasoning, modelling, procedural thinking, and future pedagogical practice. Understanding how pre-service mathematics teachers experience programming difficulties is therefore important not only for improving programming instruction, but also for informing the design of teacher-education curricula that seek to integrate computational thinking, mathematical modelling, and digital problem-solving practices into future teaching.

The literature also suggests that programming difficulty is sensitive to instructional design. When teaching emphasizes syntax in relative isolation and provides insufficient guided application, learners may understand examples in principle while still struggling to construct, test, and debug solutions on their own. Worked examples, tracing activities, deliberate debugging practice, smaller authentic tasks, and repeated guided application can help reduce this gap between explanation and execution (Hundhausen et al., 2002; Krpan et al., 2015; Lister et al., 2004; Ou et al., 2023). Evidence from online programming environments further indicates that task difficulty and sequencing can shape programming-learning outcomes (Wang et al., 2023). Programming difficulty should therefore be understood not only as a learner deficit, but also as a question of instructional sequencing, practice opportunities, feedback structures, and technological support.

This design-sensitive view is particularly relevant in teacher education. Future teachers who learn programming are not simply acquiring another university subject; they may also be developing a resource for later classroom practice. In mathematics teacher education, programming has been discussed as a way to support modelling, representation, procedural reasoning, and conceptual exploration rather than as a purely technical end in itself (Brandsaeter & Berge, 2025; Munthe & Naalsund, 2024). For pre-service mathematics teachers, programming is therefore both content to be learned and a potential pedagogical tool.

The distinction between conceptual and application/debugging-related difficulty is central to the present study. If the main bottleneck lies in access to basic concepts, instructional responses should prioritize explanation, sequencing, and representational clarity. If the bottleneck lies more strongly in transfer, debugging, and coping with unsuccessful attempts, then different responses are needed: denser practice, guided application, explicit error interpretation, and tasks that connect programming to meaningful disciplinary activity.

The study adopts this design-oriented stance cautiously. It does not test a formal theory or a full motivational model. Motivational orientation is treated as a brief descriptive index, with competence, value, and motivational-support literature used to interpret why challenge and positive orientation may coexist (Guay, 2022; Keller, 2010; Ryan & Deci, 2017, 2020). Support is approached through concrete indicators rather than assumed to form a latent construct.

Within Algeria and the wider Arab context, research on programming learning in teacher education remains limited, with existing work focusing more often on ICT integration, educational technology, school-level initiatives, or institutional challenges (Al-Buainain, 2024; Bensafa, 2015). This study therefore examines whether the stronger reported bottlenecks lie in conceptual access or in application, debugging, and sustained engagement.

The present study also recognizes that some reported programming difficulties may overlap with broader cognitive processes. In particular, the item referring to perceived difficulties in logical problem solving should not be interpreted as a purely programming-specific construct. However, introductory programming requires learners to analyse problems, decompose tasks into smaller steps, formulate algorithmic procedures, and evaluate the correctness of solutions. Difficulties in logical problem solving may therefore become visible during programming activities, especially when students attempt to implement, test, and debug a code. For this reason, the item was included as a potential indicator of programming-related difficulty while acknowledging that it may also reflect broader perceptions of the reasoning ability beyond programming itself.

3. Method

3.1. Setting and participants

The study was conducted at one Algerian Higher School for Teachers. The accessible population comprised 282 pre-service mathematics teachers, including 185 second-year students and 97 third-year students. By using convenience sampling, all students who were present during the data-collection period were invited to participate. Data were collected in June 2025, near the end of the 2024–2025 academic year and before final examinations. A total of 247 valid questionnaires were collected, corresponding to a response rate of 87.6%. The final sample included 150 second-year students (60.7%) and 97 third-year students (39.3%).

Programming was encountered across both years of the programme, but in different programme contexts. In the second year, students took a Computer Science course whose practical component focused on algorithm writing and implementation in Fortran. Whereas, in the third year, students encountered programming within Numerical Analysis through the implementation of numerical methods in Fortran. Accordingly, the second-year versus third-year comparison was treated as a programme context indicator reflecting different curricular experiences rather than a developmental year effect.

In this programme, Fortran should be understood primarily as a curricular and institutional choice. It appears in both the second-year computer science course and numerical

analysis, functioning as a shared computational medium linking algorithmic reasoning, procedural implementation, and mathematically oriented computation. The study does not assume that the language choice alone explains students' reported learning difficulties; rather, Fortran is treated as one element of the local instructional environment. Beyond its institutional presence in the curriculum, Fortran has educational relevance in mathematics-oriented programmes because of its long-standing association with scientific computing, numerical methods, and mathematical modelling. Its use enables students to connect algorithmic reasoning with computational approaches frequently encountered in applied mathematics. In this sense, the language functions not only as a programming tool but also as a medium through which mathematical procedures can be translated into executable computational processes. By the time of data collection, some second-year practical groups had not yet reached the Fortran programming component because of time constraints, syllabus density, and delays in practical coverage, and had therefore remained focused mainly on algorithms.

3.2. Instrument

The data were collected by using a structured questionnaire developed and administered in Arabic. The questionnaire was a literature-informed, context-adapted instrument rather than a previously standardized scale. Its domains and item content reflected recurring themes in programming-education research, including conceptual difficulty, application/debugging difficulty, support conditions, and learner orientation, and were tailored to the participating teacher-education context. The adaptation was also informed by long-term teaching experience in the same programme. Before administration, the Arabic version was reviewed by three experts in computer education and teaching methods in order to support face and content appropriateness. No separate pilot psychometric testing, cognitive interviewing, or external validation was conducted before the main administration.

The questionnaire included background items, 13 five-point Likert-type items used in the measurement analyses, and additional descriptive, preference, and open-ended items. The Likert-type items comprised seven difficulty items, three support items, and three motivation items. The item content is reported in Table 2 through English content labels corresponding to the Arabic questionnaire items used in data collection. The questionnaire used for data collection was in Arabic.

Since the questionnaire was locally adapted for exploratory use, the measurement analyses were treated as supportive rather than instrument-validating. Internal consistency estimates, inter-item correlations, and exploratory factor analyses were used to organize the observed response pattern, not to establish a finalized psychometric instrument. Participant-level means were computed for conceptual difficulty, application/debugging-related difficulty, and motivational orientation. The support items were retained as separate indicators because their internal coherence was too weak to justify a composite score.

3.3. Data analysis

Likert responses were coded from '1' to '5' by using the original Arabic response labels. Binary variables were coded as follows: gender ('0' = male, '1' = female), programme context ('0' = second year, '1' = third year), prior programming experience ('0' = no, '1' = yes), and self-learning of programming ('0' = no, '1' = yes). In interpretation, however, the second-year versus third-year variable was treated as a programme context indicator rather than as a developmental measure.

Descriptive statistics were computed for individual items, support indicators, and exploratory score summaries. As the main substantive focus was reported programming difficulty, the measurement analysis was organized in two layers. First, the seven difficulty-related items were examined through item-level descriptives, inter-item correlations, and a restricted exploratory factor analysis in order to determine whether they were better represented as one broad difficulty score or as two interpretable patterns. Second, a broader questionnaire-level exploratory factor analysis of all 13 Likert-type items was retained only as a secondary descriptive mapping exercise.

For the restricted difficulty-focused analysis, factors were extracted by using principal-axis factoring with oblimin rotation. The number of factors retained was judged by using the scree pattern, eigenvalues, and interpretability of the rotated solution. Based on these exploratory analyses, participant-level means were computed for conceptual difficulty and application/debugging-related difficulty as cautious descriptive summaries of the seven difficulty items. A brief motivational orientation index was also computed for descriptive purposes. These summaries were used for group comparisons and supplementary association analyses, and not as validated scales.

Welch's independent-samples t-tests were used as preliminary bivariate checks of group differences, while HC3-robust regression models were used as the main inferential analyses because they allowed simultaneous adjustment for the gender, programme context, and prior programming experience. To examine whether these patterns remained after simultaneous adjustment, multiple linear regression models with HC3 robust standard errors were estimated for the same three outcomes. Prior programming experience was retained in the main models as the primary indicator of pre-entry exposure, whereas self-learning of programming was examined in sensitivity models because the two variables were strongly related but not identical.

Missing data were handled by listwise deletion for the EFA and regression analyses, whereas descriptive statistics and correlations were based on all available responses for the variables involved. Open-ended responses were analysed in Arabic through brief author-led inductive coding in two passes. Due to the fact that the qualitative material was used only to contextualize the survey findings, coding was conducted by the author alone, no intercoder reliability coefficient was computed, and selected short excerpts were translated into English for illustrative reporting.

3.4. Ethical considerations

The study involved an anonymous, minimal-risk questionnaire administered with institutional permission in a routine higher-education setting. No formal ethics committee operated at the institution at the time; therefore, no approval number or exemption document was available. Participation was voluntary, and the students were informed that participation or non-participation would not affect grades, instructor relations, or access to learning activities. Completion of the questionnaire was taken as informed consent. No identifying information was collected, only anonymized data were analysed, and all the respondents were adults.

4. Results

4.1. Participant profile and programming background

The sample was predominantly female (168 of 247, 68.0%) and had a mean observed age of 20.56 years (SD = 1.07; range = 18–24; age available for 230 respondents). The final sample included 150 second-year students (60.7%) and 97 third-year students (39.3%). Prior programming experience was uncommon: 46 students (18.6%) reported prior experience, and the same number reported self-learning of programming. Cross-tabulation showed that these two variables were related but not identical: 186 respondents reported neither prior experience nor self-learning, 31 reported both, 15 reported prior experience without self-learning, and 15 reported self-learning without prior experience.

Among the respondents who answered the Fortran usefulness item with either ‘yes’ or ‘no’ (n = 135), 76.3% considered Fortran useful. By contrast, 78.7% of valid responses stated that the current instruction was judged as insufficient. This suggests that many students valued Fortran while also perceiving the current instructional provision as inadequate.

Table 1. *Sample characteristics and programming background*

Variable	Category	n	%
Gender	Male	79	32.0
	Female	168	68.0
Academic year	Second year	150	60.7
	Third year	97	39.3
Prior programming experience	No	201	81.4
	Yes	46	18.6
Self-learning of programming	No	201	81.4
	Yes	46	18.6
Fortran usefulness, valid yes/no responses only	No	32	23.7
	Yes	103	76.3
Current instruction sufficient?	No	188	78.7
	Yes	51	21.3

Note. Percentages are based on valid responses. Fortran usefulness excludes “I did not study Fortran” responses.

4.2. Item-level patterns, support indicators, and design preferences

The strongest reported difficulties were concentrated on the application/debugging-related side of programming learning. Debugging had the highest mean (M = 3.67), followed closely by frustration during coding (M = 3.64), theory-to-practice transfer (M = 3.63), and lack of logical problem-solving skills (M = 3.63). The conceptual items were lower overall: arrays/tables (M = 3.37), loops (M = 3.24), and conditional constructs (M = 2.62).

Support indicators were moderate, ranging from M = 3.02 for learning resources to M = 3.22 for instructor support. Motivational orientation was consistently high, with perceived professional benefit being the highest (M = 4.34), followed by confidence (M = 4.14) and enthusiasm (M = 4.13). Thus, students reported substantial difficulty while still expressing positive orientation toward learning programming.

Table 2. *Item-level descriptive statistics for difficulty, support indicators, and motivational orientation*

Domain	Item	Mean	SD	N
Conceptual difficulty	Conditional constructs	2.62	1.22	245
	Loops	3.24	1.30	245
	Arrays/tables	3.37	1.22	243
Application/debugging-related difficulty	Lack of logical problem-solving skills	3.63	1.25	241
	Theory-to-practice transfer	3.63	1.09	246
	Debugging	3.67	1.09	245
	Frustration during coding	3.64	1.17	241
Support indicator	Support from instructors	3.22	1.10	246
	Discussion with peers	3.15	1.13	244
	Learning resources are sufficient	3.02	1.25	245
Motivational orientation	Confidence	4.14	1.00	246
	Enthusiasm	4.13	0.99	246
	Perceived professional benefit	4.34	0.98	246

Note. Means are based on a 1–5 response scale. Higher values indicate stronger agreement with each statement.

Table 3. *Design preferences reported by students*

Panel	Response option	Frequency	Percentage
Preferred learning method	Individual exercises	100	40.5
	Educational videos	84	34.0
	Practical projects	68	27.5
	Other	67	27.1
Suggested improvement	More practice time	189	76.5
	More practical exercises	164	66.4
	Better interaction with instructors	147	59.5
	Modern languages	146	59.1
	More projects	124	50.2
	Additional self-study materials	112	45.3

Note. Percentages are based on the full sample (N = 247). As the respondents could select more than one option, percentages do not sum up to 100% within each panel.

Students' design preferences reinforced this pattern. They most often requested more practice time (76.5%), more practical exercises (66.4%), better interaction with instructors (59.5%), and more modern languages or tools (59.1%). The preferred learning methods were more dispersed, with individual exercises (40.5%) and educational videos (34.0%) selected more often than practical projects (27.5%).

4.3. Measurement analysis: exploratory score summaries and factor structure

The measurement analysis was used to examine whether the seven difficulty-related items were better represented as one broad difficulty score or as a distinction between conceptual difficulty and application/debugging-related difficulty.

The exploratory score summaries supported this distinction. The broad difficulty summary had acceptable internal consistency ($\alpha = .76$), but the two-pattern interpretation was more informative: conceptual difficulty had a mean of 3.08 (SD = 1.04, $\alpha = .79$), whereas application/debugging-related difficulty had a higher mean of 3.64 (SD = 0.82, $\alpha = .66$). Motivational orientation was high (M = 4.20, SD = 0.77, $\alpha = .67$). The support items were not retained as a composite because their internal consistency was too low ($\alpha = .22$).

Table 4. *Exploratory score summaries and internal consistency evidence*

Score summary / index	Items	N	Mean	SD	Cronbach's alpha	Interpretive status
Broad difficulty summary	7	247	3.40	0.78	.76	Descriptive benchmark only
Conceptual difficulty	3	247	3.08	1.04	.79	Exploratory score summary
Application/debugging-related difficulty	4	247	3.64	0.82	.66	Exploratory score summary
Brief motivational orientation index	3	246	4.20	0.77	.67	Brief descriptive index
Support indicators	3	—	—	—	.22	Not retained as a composite

Note. Scores were computed as mean item responses. Support indicators were not combined because internal consistency was low.

The restricted seven-item exploratory factor analysis provided the clearest measurement signal. The three conceptual items loaded primarily on one factor, whereas theory-to-practice transfer, debugging, frustration during coding, and self-perceived logical problem-solving difficulty loaded primarily on a second factor. Sampling adequacy was acceptable (KMO = 0.76), Bartlett's test was significant, $\chi^2(21) = 395.87$, $p<.001$, and the first two eigenvalues were 2.94 and 1.20.

A broader 13-item questionnaire-level exploratory factor analysis was also inspected as a secondary descriptive mapping exercise. It showed interpretable clustering for conceptual difficulty, motivational orientation, and application/debugging-related difficulty,

whereas the support indicators showed weak communalities and no coherent retained factor. Accordingly, support was reported separately, and motivation was retained only as a brief descriptive index.

Taken together, the measurement evidence supports a distinction between conceptual difficulty and application/debugging-related difficulty, while motivation is treated as a brief descriptive index and support as separate indicators.

Table 5. *Restricted exploratory factor analysis of the seven difficulty items*

Item	F1: Conceptual difficulty	F2: Application/ debugging-related difficulty	h^2
Conditional constructs	0.84	-0.10	0.72
Loops	0.87	0.06	0.76
Arrays/tables	0.77	0.06	0.60
Lack of logical problem-solving skills	0.21	0.65	0.46
Theory-to-practice transfer	0.02	0.76	0.58
Debugging	-0.09	0.72	0.53
Frustration during coding	-0.07	0.67	0.46

Note. Principal-axis factoring with oblimin rotation; n = 228 complete cases. All loadings are shown. Future validation should examine the structure in independent samples.

4.4. Group differences and adjusted associations

The exploratory score-summary structure revealed a differentiated pattern. There were no statistically significant gender differences for conceptual difficulty, application/debugging-related difficulty, or motivational orientation.

The preliminary Welch tests showed the same substantive pattern as the adjusted models reported in Table 6; therefore, the text summarizes the bivariate findings while Table 6 presents the main adjusted estimates.

The programme context and prior-experience differences were selective. Conceptual difficulty was higher in the second-year programme context and among students without prior programming experience, whereas application/debugging-related difficulty showed no statistically significant bivariate difference. Motivational orientation was higher in the second-year programme context and among students with prior programming experience.

In the HC3-robust models, the same pattern remained: conceptual difficulty was lower in the third-year programme context and among students with prior experience, the application/debugging-related model explained little variance and had no significant predictors, and motivational orientation was lower in the third-year programme context but higher among students with prior experience. Sensitivity models replacing prior programming experience with self-learning yielded the same substantive pattern.

Table 6. *HC3-robust regression models examining associations with exploratory score summaries*

Outcome	Covariate	B	SE (HC3)	95% CI	*p*	R^2	N
Conceptual difficulty	Female	-0.14	0.13	[-0.39, 0.12]	0.297	0.186	247
	Third-year programme context	-0.83	0.12	[-1.07, -0.58]	< .001	0.186	247
	Prior experience	-0.50	0.16	[-0.82, -0.19]	0.002	0.186	247
Application/ debugging-related difficulty	Female	0.12	0.12	[-0.12, 0.36]	0.317	0.018	247
	Third-year programme context	-0.12	0.11	[-0.33, 0.09]	0.260	0.018	247
	Prior experience	-0.21	0.13	[-0.46, 0.04]	0.106	0.018	247
Motivational orientation	Female	0.09	0.11	[-0.13, 0.30]	0.423	0.072	246
	Third-year programme context	-0.26	0.11	[-0.46, -0.05]	0.015	0.072	246
	Prior experience	0.39	0.10	[0.19, 0.60]	< .001	0.072	246

Note. Models included gender, programme context, and prior programming experience. Programme context should not be interpreted as a pure developmental year effect.

4.5. Correlations, categorical associations, and qualitative evidence

Conceptual difficulty and application/debugging-related difficulty were moderately positively related ($r = 0.433$, $p<.001$). Neither difficulty subscale was significantly associated with motivational orientation: conceptual difficulty, $r = -0.019$, $p = 0.761$; application/debugging-related difficulty, $r = -0.056$, $p = 0.383$. Prior programming experience was not significantly associated with gender, $\chi^2(1) = 0.18$, $p = 0.671$, or academic year, $\chi^2(1) = 0.27$, $p = 0.600$, but was strongly associated with self-learning, $\chi^2(1) = 84.80$, $p<.001$, Cramer's $V = 0.586$.

Out of the 247 respondents, 120 provided non-empty open-ended answers. Their comments echoed the quantitative pattern by highlighting the need for more curricular time, stronger guided application, authentic projects or real problems, modern tools, and stronger instructional support.

To make this supplementary evidence more transparent, Table 7 presents short translated excerpts from the open-ended responses. These excerpts are used only to contextualize the survey findings and are not treated as an independent qualitative strand or as evidence of formal triangulation.

Table 7. *Illustrative translated excerpts*

Theme	Illustrative translated excerpt	Instructional implication
More curricular time and continuity	"Programming should be taught as a regular course in the curriculum rather than as practical work only once every 15 days; otherwise, students forget what they learn."	Provide more continuous exposure and avoid long gaps between practical sessions.
More guided application and exercises	"I want more attention to the practical side than to the theoretical side."	Increase guided exercises and scaffolded implementation tasks.
Authentic projects and real problems	"There should be application domains or solutions to real problems."	Use meaningful, mathematics-linked, or authentic programming tasks.
Modern tools and stronger support	"Add more modern languages than Fortran, such as C++, and provide enough support and time for students to study programming."	Combine tool/language updating with stronger instructional support.

Note. The original comments were written in Arabic and translated into English for reporting. The excerpts are illustrative, anonymized, and used only as supplementary contextual evidence.

5. Discussion

The findings should be read as a context-bound but internationally relevant case of programming learning in non-specialist teacher education. In this sample, reported programming difficulty was more informatively described through item-level evidence and a distinction between conceptual difficulty and application/debugging-related difficulty than through one broad undifferentiated score. The strongest pressure points concerned theory-to-practice transfer, debugging, frustration during coding, and self-perceived weakness in logical problem solving. This pattern is broadly consistent with previous programming-education research indicating that novice learners often experience greater difficulty when moving from conceptual understanding to independent implementation and debugging (Robins et al., 2003; Lister et al., 2004; Luxton-Reilly et al., 2018). Rather than suggesting a lack of conceptual exposure alone, the findings point to the challenge of translating knowledge into executable procedures and managing the iterative process of testing and correcting a code. From an instructional perspective, this reinforces the importance of learning environments that provide opportunities for guided application, error diagnosis, and structured practice beyond the presentation of programming concepts.

A second key result is that conceptual difficulty and application/debugging-related difficulty did not behave identically. Conceptual difficulty showed clearer differences across the second-year and third-year programme contexts and by prior programming experience, whereas application/debugging-related difficulty showed less differentiation across the compared groups after adjustment. This pattern may reflect differences in curricular exposure rather than developmental change alone. In this context, some reported programming difficulty may be linked not only to the intrinsic challenge of programming, but also to limited opportunities to engage fully with the programming language because of time constraints and syllabus density. At the same time, the weaker differen-

tiation of application/debugging-related difficulty suggests that applying, testing, and repairing a code may require sustained instructional support across programme contexts. Due to the study being cross-sectional, these contrasts should not be interpreted as direct developmental change.

The distinction between these two patterns is also educationally meaningful. Much of the literature on introductory programming treats programming difficulty as a relatively unified phenomenon. The present findings suggest that, in mathematics teacher-education settings, difficulties related to conceptual understanding may not necessarily coincide with difficulties related to implementation, debugging, and sustained problem solving. This distinction may help teacher educators identify more targeted instructional responses rather than treating all programming difficulties as manifestations of the same underlying challenge.

The motivation findings should be interpreted more cautiously than the difficulty findings because motivation was represented only through a brief three-item descriptive index. Even so, the coexistence of high perceived value and substantial reported difficulty suggests that difficulty in this context did not simply reflect disengagement. In professionally oriented programmes, students may continue to value programming while still needing stronger scaffolding for application. Mathematics-linked tasks and small projects may therefore help preserve perceived relevance while supporting implementation.

The support findings should also be interpreted at the indicator level rather than as evidence of a unified support construct, because the three support items did not show sufficient internal coherence to justify combination. Nevertheless, the student-preference and open-ended evidence gives these indicators practical design meaning: students repeatedly asked for more practice time, more exercises, stronger interaction, projects, and more contemporary tools. The Fortran finding points in the same direction: many students viewed the language as useful, but most judged the current instruction insufficient. Taken together, the data suggest that the stronger issue lies less in the language choice alone than in continuity, guided application, and instructional interaction.

These implications should be interpreted in line with the study design. The results are based on self-reports from a single site and identify perceived instructional pressure points rather than objective programming competence. Future research should combine self-report evidence with performance-based measures, course achievement data, classroom observation, or think-aloud problem-solving protocols.

5.1. Contribution to teacher education and programming education

The contribution of this study lies in extending programming-education research to a teacher-education context that remains underrepresented in the international literature. While much of the existing evidence originates from computer science or engineering programmes, the present study focuses on pre-service mathematics teachers learning programming within a broader professional curriculum. The findings suggest that application and debugging challenges may represent more salient instructional pressure

points than conceptual understanding alone. By distinguishing between conceptual difficulty and application/debugging-related difficulty, the study provides a more differentiated view of programming learning challenges and highlights the importance of instructional design in supporting future teachers' engagement with computational approaches. The study also contributes empirical evidence from an Algerian and Arabic-speaking educational context, by adding geographical and contextual diversity to the programming-education literature.

6. Limitations

Several limitations should be acknowledged. First, the study was conducted in a single institution and used convenience sampling; the findings should therefore be interpreted as context-specific rather than broadly representative. Second, the evidence is based primarily on self-report data and does not include direct performance measures of programming competence, classroom observation, or intervention data. The results should therefore be read as identifying *perceived* instructional pressure points rather than measuring *objective* programming ability.

Third, the questionnaire was literature-informed and author-adapted in Arabic for a specific local context. Although it was reviewed by three experts for face and content appropriateness, it was not a previously standardized scale and was not subjected to separate pilot psychometric testing, cognitive interviewing, or external validation before the main administration. Accordingly, the retained summaries should be interpreted as exploratory and context-specific rather than as validated constructs. The support items did not justify a composite score, and the application/debugging-related grouping should be interpreted in light of the exploratory nature of the analysis.

Fourth, the same sample was used both to derive the exploratory score summaries and to examine their group differences and adjusted associations. Although a broader 13-item questionnaire-level exploratory factor analysis was inspected, the clearest measurement signal came from the restricted analysis of the seven difficulty items. The broader analysis was therefore used only as a descriptive mapping exercise, and not as a basis for strong construct-level inference.

Fifth, the qualitative component was coded by the author alone and used only as supplementary contextual evidence. No intercoder reliability coefficient was computed, and the illustrative quotations were translated from Arabic into English, which adds an additional interpretive step.

Finally, because the study is cross-sectional, differences by prior programming experience and by second-year versus third-year programme context should not be interpreted causally. The two groups of different study years differed in their immediate curricular setting, and some second-year practical groups had not yet completed the planned Fortran component because of time constraints, syllabus density, and delays in practical coverage. The observed group differences therefore cannot be disentangled from uneven exposure to the programming content and should not be interpreted as a pure extra-year effect.

7. Conclusion

This study used one Algerian teacher-education setting to examine how pre-service mathematics teachers report programming learning difficulties in a non-specialist curriculum. The findings suggest that the most salient difficulties were concentrated in theory-to-practice transfer, debugging, and frustration during coding rather than in basic concepts alone. Conceptual difficulty differed more clearly by the programme context and prior programming experience, whereas application/debugging-related difficulty showed no clear adjusted group differences. This contrast should be interpreted as a programme context pattern rather than as a developmental year effect.

The main contribution of the study is to highlight an instructional design issue in non-specialist programming education: conceptual explanation alone may be insufficient if learners are not also supported in applying, testing, and repairing the code. The findings point toward denser practice, guided application, explicit debugging support, and mathematics-linked tasks in teacher education, particularly in non-specialist programming contexts where implementation challenges may outweigh conceptual difficulties. As a single-site self-report study, these findings should be replicated through multi-site, performance-linked, and instrument-development research.

Declaration of Generative AI Use

Generative AI tools, including *ChatGPT*, were used only for language editing, formatting support, and manuscript-readiness checking. They were not used to generate data, conduct statistical analyses, make analytic decisions, or replace the author's scholarly interpretation. The author reviewed and verified the final manuscript and takes full responsibility for its content.

Declarations

Acknowledgments. *Not applicable.*

Funding. *This research received no specific grant from any funding agency.*

Ethics statement. *Formal ethical approval was not available because the participating Higher School for Teachers did not have a dedicated ethics committee for educational research at the time of data collection. The study was conducted with institutional permission and involved an anonymous, minimal-risk questionnaire completed by adult respondents.*

Conflict of interest. *The author declares no conflict of interest.*

Informed consent. *Completion of the questionnaire was taken as informed consent. Participation was voluntary, responses were anonymous, and no identifying information was collected.*

Data availability. *The anonymized data are available from the author upon reasonable request.*

References

Al-Buainain, M. R. (2024). The effect of training based on the programming language (Scratch/Python) in developing mental habits and reducing developmental learning difficulties among primary school students. *Journal of Studies in Orthophonia and Neuropsychology*, 8(1), 31–45. https://www.asjp.cerist.dz/en/article/254824

Abdelkader, B. (2015). ICT in Algerian education: Current trends and future challenges. *Arab World English Journal (AWEJ),* (Special Issue on CALL No. 2), 226–234. https://ssrn.com/abstract=2843992

Brandsæter, A., & Berge, R. L. (2025). Promoting mathematical competence development through programming activities. *Educational Studies in Mathematics*, *119*(2), 225–247. https://doi.org/10.1007/s10649-024-10380-y

Derus, S. R. M., & Mohamad Ali, A. Z. (2012). *Difficulties in learning programming: Views of students*. In *Proceedings of the 1st International Conference on Current Issues in Education (ICCIE 2012)* (pp. 74–79). Yogyakarta State University. https://doi.org/10.13140/2.1.1055.7441

Guay, F. (2022). Applying Self-Determination Theory to Education: Regulations Types, Psychological Needs, and Autonomy Supporting Behaviors. *Canadian Journal of School Psychology*, *37*(1), 75–92. https://doi.org/10.1177/08295735211055355

Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages & Computing*, *13*(3), 259–290. https://doi.org/10.1006/jvlc.2002.0237

Jenkins, T. (2002). *On the difficulty of learning to program*. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences* (pp. 53–58). Loughborough University.

Keller, J. M. (2010). *Motivational design for learning and performance: The ARCS model approach.* Springer. http://dx.doi.org/10.1007/978-1-4419-1250-3

Krpan, D., Mladenović, S., & Rosić, M. (2015). Undergraduate Programming Courses, Students' Perception and Success. *Procedia - Social and Behavioral Sciences*, *174*, 3868–3872. https://doi.org/10.1016/j.sbspro.2015.01.1126

Lister, R., Adams, S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Mostrom, J. E., Sanders, K., Seppala, O., Simon, B., & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin, 36*(4), 119–150. https://doi.org/10.1145/1041624.1041673

Luxton-Reilly, A., Simon, B., 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*. In *ITiCSE '18 Companion: Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (pp. 55–106). ACM. https://doi.org/10.1145/3293881.3295779

Munthe, M., Naalsund, M. (2024). Designing Mathematical Programming Problems. *Digital Experiences in Mathematics Education*, *10*(2), 260–286. https://doi.org/10.1007/s40751-024-00143-y

Ngadengon, Z., Subramaniam, T. S., Yasak, Z., Ideris, N. A., & Mohd Ramly, Z. (2025). Evaluating the Difficulties in Programming Learning: Insights from Polytechnic Students. *International Journal of Academic Research in Progressive Education and Development*, *14*(1), 1051–1065. https://doi.org/10.6007/IJARPED/v14-i1/24518

Ou, Q., Liang, W., He, Z., Liu, X., Yang, R., & Wu, X. (2023). Investigation and analysis of the current situation of programming education in primary and secondary schools. Heliyon, 9, Article e15530. https://doi.org/10.1016/j.heliyon.2023.e15530

Putri, T. T. A., Yahaya, W. A. J. W., Sriadhi, S., & Alie, M. F. (2024). *A preliminary study: The cause of university students' difficulties in programming subject*. In *Proceedings of the 5th International Conference on Innovation in Education, Science, and Culture (ICIESC 2023)*. EAI. https://doi.org/10.4108/eai.24-10-2023.2342079

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

Ryan, R. M., Deci, E. L. (2017). *Self-determination theory: Basic psychological needs in motivation, development, and wellness.* Guilford Press.

Ryan, R. M., Deci, E. L. (2020). Intrinsic and extrinsic motivation from a self-determination theory perspective: Definitions, theory, practices, and future directions. *Contemporary Educational Psychology*, *61*, Article 101860. https://doi.org/10.1016/j.cedpsych.2020.101860

Wang, J., Lin, P., Tang, Z., & Chen, S. (2023). How problem difficulty and order influence programming education outcomes in online judge systems. *Heliyon, 9*(11), Article e20947. https://doi.org/10.1016/j.heliyon.2023.e20947