The Nordic Prior Knowledge Test in Programming: Motivation, Development and Preliminary Results

Sondre S. Bolland[1], Andreas Haraldsrud[2], Siri M. Jensen[2], Filip Strömbäck[3], Arne Styve[4], Erlend Tøssebro[5], Eirik Valseth[6], and Torstein J. F. Strømme[1]

[1] University of Bergen
[2] University of Oslo
[3] Linköping University
[4] The Norwegian University of Science and Technology
[5] University of Stavanger
[6] Norwegian University of Life Sciences

Abstract. With recent updates to Norway's national curriculum, computational thinking and programming has become a core part of the K-12 education, leading to an influx of students entering higher education with prior programming experience. This shift has the potential to impact the teaching of ICT at universities, as foundational knowledge could allow for the introduction of more advanced topics earlier. However, the quality of programming education varies, making it essential to assess students' prior knowledge.

To address this need, the *Nordic Prior Knowledge Test in Programming* was designed to assess incoming students' programming proficiency in the basic elements of the introductory programming course (CS1). This study details the rationale for assessing incoming students, the development and content of the test, and the preliminary results from its 2024 administration.

The test was completed by 3,038 students (2,661 after data pruning) across eight higher education institutions in Norway. Results indicate a mean score of 39.9%, with a significantly higher performance among students exposed to the new curricular model (50%). Despite these gains, a substantial proportion of students scored at the lower end of the scale, highlighting the ongoing need for foundational programming instruction.

Although most students will benefit from completing the standard CS1 course, a notable subset of students achieved high scores (14.7% scoring above 90%), suggesting the potential value of accelerated or alternative learning pathways, such as an advanced CS1 course or direct progression to CS2.

Keywords: Prior Knowledge · Assessment · Introductory Programming · CS1

1 Introduction

The integration of programming and computational thinking into educational curricula is becoming increasingly widespread, with several countries incorporating computer science as a core element in elementary and secondary school (K-12) education. In Norway programming was made part of the national K-12 curriculum in 2020 [34]. This shift suggests that students entering higher education may already possess a foundational understanding of introductory programming concepts. This development may have profound implications for the teaching of computer science subjects at universities and university colleges, potentially enabling instructors to introduce more advanced topics in the very first semester.

However, the quality of K-12 programming instruction has exhibited considerable variation since the educational reform [14, 39]. One notable factor contributing to this variability is the differing levels of programming expertise among teachers, ranging from those with formal training in computer science didactics to those with minimal or no programming experience. As a result, assessing the prior programming knowledge of incoming students is critical for understanding and mitigating the disparities in educational outcomes.

The *Nordic Prior Knowledge Test in Programming* has been developed as an instrument to assess students' proficiency in programming. This test is designed to cover key components of the curriculum objectives typically addressed in the introductory programming course (CS1) [16]. Its primary aim is to identify misconceptions held by incoming students, enabling instructors to tailor and adapt their courses more effectively to match the students' existing skill levels and understanding.

1.1 Research Questions & Structure

The Nordic Prior Knowledge Test in Programming provides a rich dataset designed to address a broad range of research questions. In this study, we focus on answering the following:

- **RQ1:** How has the introduction of computational thinking and programming into the national K-12 curriculum impacted the programming competency of students entering higher education in Norway?
- **RQ2:** Is there still a need to conduct the CS1 course for incoming higher education students?

Additional research questions, to be addressed in future publications, include:

- **RQ3:** Are there significant differences in prior programming knowledge based on students' demographics and educational backgrounds?
- **RQ4:** How familiar are the students with specific programming topics, and what are the common misconceptions?
- **RQ5:** How has the programming competency of incoming students changed since the 2023 iteration of the Nordic Prior Knowledge Test in Programming?

This paper presents preliminary findings from the 2024 iteration of the test. We begin by providing an overview of the K-12 educational context in Norway, along with a review of previous assessments of programming knowledge. The methodology section details the structure and content of the test, as well as participant demographics. In the results section, we report the overall performance of students, comparing outcomes between the new and previous curricular models. The article concludes with an interpretation of these results and a discussion of their implications for programming instruction in higher education.

2 K-12 Computer Science Education

The integration of computer science into K-12 education has gained increasing recognition as vital in todays technology-driven society. Over the past 15 years, several countries have introduced reforms aimed at incorporating computational thinking and programming as core elements of their curricula, including New Zealand [4], the United Kingdom [10], Finland [6], Sweden [38], and Germany [18].

2.1 Integration of Programming in Norwegian K-12

The educational reform "Kunnskapsløftet 2020" introduced computational thinking and programming as core components of the Norwegian K-12 curriculum in 2020 [34]. Programming was incorporated into various subjects, with a particularly strong emphasis in mathematics, where it serves primarily as a tool for solving mathematical problems. By the end of 10th grade, students are expected to achieve the following learning outcome (translated from Norwegian) [27]:

"Explore mathematical properties and relationships using programming."

At the upper secondary school level, both programming and computational thinking are explicitly stated in the objectives within the mathematics courses [28–32]. Key learning outcomes for these courses include (translated from Norwegian):

> *"Formulate and solve problems using computational thinking, problem-solving strategies, digital tools, and programming."* (Mathematics T [30])
>
> *"Explore recursive relationships using programming and present your own approaches."* (Mathematics R and S [31,32])
>
> *"Develop algorithms for numerically calculating integrals and use programming to implement the algorithms"* (Mathematics R [31])

Furthermore, upper secondary school students may (depending on whether their school offer them) have access to these elective courses focused on computing:

– Information Technology 1 [26]
– Information Technology 2 [26]
– Programming and Modelling X [33]

Programming and Modelling X emphasizes the development, visualization and interpretation of mathematical and scientific models of dynamic systems. *Information Technology 1* focuses on the role of technology and programming in society, covering data representation, website development, and the societal implications of technology. *Information Technology 2* addresses more advanced topics, typically found in a first-year computer science curriculum, such as program design, debugging, reusable code, object-oriented programming, collaboration tools, data management, and data analysis.

Recently, programming has also been included in national exams for secondary school students. For instance, the course Mathematics R2[7] [31] included the following task in the spring 2024 exam [35] to be solved without the use of any digital aids:

A student has written the following code:

```
n = 0
S = 0

while S <= 200:
    n = n + 1
    S = S + 4*n - 2

print(n)
```

a) Explain what the student is trying to find out.
b) What value is printed when the program is run?

This task requires students to demonstrate an understanding of key programming concepts such as loops, conditionals, and variable (re-)assignments. Through such tasks, students are also required to infer mathematical or scientific information from code, and vice versa. However, even though these concepts are assessed in school exams, it does not guarantee that all students have mastered them.

2.2 The Teacher Crisis

The level of programming competency among teachers varies greatly, likely affecting the educational outcomes of their students. Prior to its implementation, teachers shared their perspectives on the upcoming educational reform, acknowledging the importance of programming competence within the educational framework while anticipating significant challenges in teaching content from a relatively unfamiliar domain [14].

Stenlund's 2021 study [39] conducted interviews with teachers revealing a broad spectrum of programming proficiency. Those with prior experience in programming were better positioned to

[7] Mathematics R1 and R2 are aimed at students preparing for STEM programs.

provide instruction aligned with computer science principles, while others remain largely unfamiliar with the subject. A frequently cited issue is the lack of a well-developed subject-specific pedagogy for programming. Although many teachers find programming to be engaging and intellectually rewarding, they also describe it as time-consuming and challenging to master. In terms of pedagogical perspectives, a notable proportion of teachers view the integration of programming into the mathematics curriculum as suboptimal, with some advocating for programming to be introduced as a standalone subject. Additionally, widespread dissatisfaction exists regarding the level of institutional support, with teachers highlighting insufficient preparation time, inadequate professional development opportunities, and a persistent lack of subject-specific pedagogical resources. Despite these difficulties, the majority of teachers interviewed remain optimistic about the long-term potential of the ongoing curriculum reform.

This variation in teacher programming competence is likely to impact students, leading to inconsistencies in programming proficiency among incoming cohorts. This underscores the need for effective assessment of students' prior knowledge to ensure appropriate instruction moving forward.

2.3 Assessment of Programming Competence

Various methods have been proposed for assessing programming competence, with numerous studies aiming to predict students' aptitude and their likelihood of succeeding in computer science. Among the most notable is the study by Dehnadi and Bornat [9], who claimed to have developed a predictive test capable of accurately forecasting the success or failure of computer science students, even prior to any exposure to programming languages. The test involved 12 tasks centered on *Assignment and Sequence*, where each task included two variable declarations followed by one, two, or three assignment statements. However, subsequent replications of their study at other institutions produced conflicting results. In these cases, the test failed to consistently predict student outcomes in introductory programming courses [7,8].

Duran et al. [11] proposed a self-evaluation instrument aimed at measuring prior programming knowledge in introductory programming courses, with an emphasis on code comprehension and reading skills. This instrument assessed several common CS1 concepts and gauged students' proficiency across various levels, ranging from total unfamiliarity with a concept to full comprehension and ability to explain the concept using abstract values. The progression included recognizing and understanding syntax, tracing programs with concrete values, and explaining code in plain English without reliance on meaningful naming conventions. The instrument demonstrated high internal consistency and showed low to moderate correlation with traditional ad-hoc measures of programming background, such as previous coursework in programming in the K-12 period, hours spent programming, and familiarity with different programming languages. Their future research is planned to focus on analyzing how students' responses correlate with other instruments for assessing programming competency, including course grades and validated tools like concept inventories (CIs).

A CI is a criterion-referenced assessment tool designed to evaluate students' precise understanding of specific concepts. CIs use carefully constructed multiple-choice questions to uncover conceptual misconceptions and help instructors refine their teaching approaches and curriculum [40]. These tools have been employed across many STEM fields, including physics [17], chemistry [25], astronomy [3], biology [1], mathematics [12], geoscience [21], and computer science [15,36,42], with notable success.

Several CIs have been developed for CS1, including FCS1 [42] and SCS1 [36], though these tools present challenges in the context of prior knowledge. In other STEM fields, CIs are commonly administered as both pre- and post-tests to track student improvement [17]. However, this method is problematic in computer science, as students historically do not begin with misconceptions, having had little prior exposure to programming. Porter et al. [37], Tew and Guzdial [41], and Webb and Becker [45] argue that misconceptions in computer science often emerge from instruction itself, making pre-test results misaligned with those gathered later in the course.

Recent studies have noted the growing importance of CIs as pre-tests in computer science, particularly as programming becomes more embedded in K-12 education [20, p. 3]. However, existing CIs may not adequately account for students' limited prior knowledge. Xie et al. [46, p. 704] caution that prior use of the SCS1 as a pre-test may have been affected by floor effects, due to the test's high difficulty level, suggesting that the SCS1 may be more suitable as a post-test. Section 7.3 discusses our ongoing work to address the pre-test limitations of current CIs.

3 Methodology

To assess the programming proficiency of incoming students, the Nordic Prior Knowledge Test in Programming integrates the approaches of Luxton-Reilly et al. [23] and Thompson et al. [43] to create an assessment focused on basic elements of the CS1 curriculum. The test was designed in Python, the primary programming language used in the Norwegian K-12 curriculum [39], and was developed in collaboration with experts from eight institutions across Norway and Sweden.

3.1 Test Content

The content of the test spans eight key programming topics that are likely encountered in K-12 education and form an important part of the CS1 curriculum. While not exhaustive of all CS1 content, these topics focus on core programming concepts. To allow for the reuse of the programming tasks in later years, only a limited selection of specific tasks is presented in this paper. Instead, we provide an overview of the topics covered to describe the programming concepts assessed by the test.

Datatypes. The datatype tasks assessed students ability to identify some of the most commonly used variable types in Python: *int*, *float*, *bool*, *str* and *list*.

Operators. These tasks tested students understanding of how operators behave when applied to variables of different types, focusing on *addition* and *concatenation*. The intent was to highlight the polymorphic nature of the + operator, and how it can be used with different datatypes.

Variables. The tasks related to variables were derived from the work of Dehnadi and Bornat [9]. Although their test could not reliably predict programming aptitude [7, 8], these variable-related tasks have proven useful in identifying student misconceptions [19]. These tasks involved two or three variable declarations followed by two or three assignment statements.

Boolean Expressions. These tasks required students to correctly evaluate boolean expressions, starting with basic comparisons like $2 > 7$ and advancing to more complex expressions involving logical operators, such as conjunction, disjunction, and negation. The tasks regarded as most difficult included abstract expressions, such as *not (True != True)*.

Conditionals. The conditional tasks asked students to predict the output of *if-statements* with embedded print statements. The complexity increased from a single *if-statement* to several *if-statements* using *if-elif-else* clauses. All conditions used in the if-statements had been assessed in isolation in the *Boolean Expressions* tasks.

Loops. Students were tested on their ability to follow the manipulation of variables (both integers and lists) within *while* and *for loops*. As above, all conditions have been tested in isolation previously.

Lists. The list tasks focused on indexing and assessing how lists are altered through various operations.

Functions. The nine function-related tasks incorporated elements from earlier topics, embedded within functions, with slight modifications to assess students' understanding of *arguments and parameters*, *return values*, and general *function semantics*.

3.2 Task Complexity

We expected that the majority of students would demonstrate a programming competency level below the learning outcomes specified for a typical CS1 course. Therefore, the test focused on the basics of the course, both in terms of the topics covered and cognitive ability.

Blooms taxonomy, a hierarchical framework for categorizing educational objectives by complexity and specificity [5], was used to guide task design. Thompson et al. [43] provided an interpretation of this (revised [2]) taxonomy for computer science assessment. At the lower levels of the taxonomy, we find cognitive processes such as *remember*, *understand*, and *apply*, which progressively increase in complexity. In the context of computer science, their interpretation of these levels included:

- identifying a particular construct in a piece of code
- recalling a conceptual definition or a process that has been previously implemented in the same context as a classroom exercise
- applying a known process, algorithm, or design pattern to a familiar problem in a new context or with different data or tools
- applying a known process, algorithm, or design pattern to an unfamiliar problem

The programming tasks in the test were designed in alignment with these interpretations. Students were required to either identify constructs or trace the execution of a program to determine its output. No task required students to write a piece of code to solve a given problem, which would align with the higher levels of Blooms taxonomy. The majority of tasks were open-ended, allowing students to provide their answers in a free-text format, while a smaller portion consisted of multiple-choice questions. All tasks were auto-graded.

3.3 Concept Dependency

To accurately identify the specific programming concepts that students struggled with or held misconceptions about, tasks needed to target individual concepts. Given that many programming concepts are interdependent, they cannot be assessed in isolation.

To address this, tasks were designed to first assess the atomic elements of programming concepts, progressively combining these elements to assess more complex topics in a comprehensive manner. This approach aligns with the methodology used by Luxton-Reilly et al. [23], which analyzed the syntactical dependencies in the Java programming language to identify atomic elements which students need to understand in order to tackle more complex structures.

Our methodology, however, adopts a broader scope. Rather than concentrating on the intricate details of a narrow set of programming concepts, we designed the tasks to capture dependencies across a broader spectrum of topics. For example, consider the following code snippet:

```
def f(n):
    while n < 10:
        n = n + 1
    return n

print(f(5))
# What does this code snippet print?
```

This task was designed to assess the students' knowledge of *Functions Semantics*. However, it also involves other concepts such as *Looping*, *Boolean Expressions*, and *Value Incrementation*. If a student fails this task, it is unclear from this task alone which concept(s) they are unfamiliar with. To resolve this issue, preceding tasks were designed to test these specific concepts individually. To illustrate this, the students are first presented with a task focused on *Value Incrementation*:

```
n = 0
n = n + 1
n = n + 1
# What is the value of n after line three has been executed?
```

A range of tasks on boolean expressions included the evaluation of $n < 10$.
The body of the function *f* is assessed in this task:

```
n = 0
while n < 10:
    n = n + 1

print(n)
# What does this code snippet print?
```

By addressing the atomic elements of programming concepts first, we can correlate students' performance on more complex tasks with their performance on these simpler tasks, enabling us to more precisely identify which concepts are contributing to errors in their responses.

However, it is important to recognize that the conceptual dependencies considered when designing this test may not encompass all the prior knowledge a student requires to solve a given task. Some atomic elements may not have been explicitly tested, or the dependencies we identified might not be directly relevant to the specific task being assessed. These dependencies will be explored further in a future publication.

3.4 Demographics

The test was administered to 3,038 students. However, to focus exclusively on participants with pre-university programming experience, the dataset was pruned to exclude those who had completed a university-level programming course. This resulted in a final sample of 2,661 students for analysis.

Institutions. The test was distributed across eight higher education institutions in Norway, shown in Table 1. While the test was also administered to students in Sweden at Linköping University (n=222), the results from that cohort will be discussed in an upcoming study.

Table 1: Number of students per participating institution.

Institution	**N**	**%**
The Norwegian University of Science and Technology	928	34.9
University of Oslo	543	20.0
University of Stavanger	334	12.6
University of Bergen	333	12.5
Norwegian University of Life Sciences	206	7.7
Kristiania University College	144	5.4
Østfold University College	95	3.5
Western Norway University of Applied Sciences	75	2.8

Graduation Years. Figure 1 illustrates the distribution of students by their upper secondary school graduation year. Notably, students who graduated in 2023 and 2024 were exposed to computational thinking and programming through the new curricular model, while those who graduated in earlier years were educated under the previous curriculum.

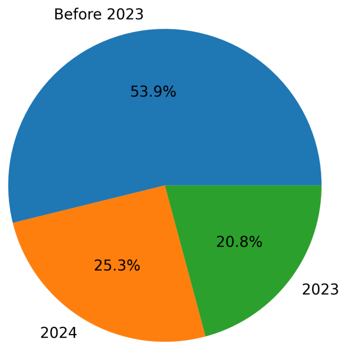

Fig. 1: Distribution of when students graduated from secondary school.

4 Results

4.1 All Students

The mean score of all participating students was 16.2 out of a possible 40.6 points (39.9%), with a standard deviation of 12.1. The distribution of scores is illustrated in Figure 2.

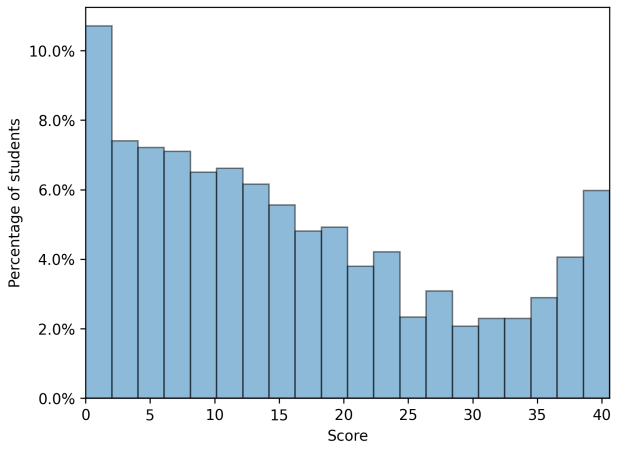

Fig. 2: Test score distribution of the all students. Max score: 40.6.

4.2 Exposure to the New Curriculum.

A Wilcoxon rank-sum test revealed a statistically significant difference between the mean scores of the 2023–2024 students and earlier cohorts ($W = 16.9$, $p < 8.9 \cdot 10^{-65}$). The 2023–2024 cohort had a mean score of 20.3 (50.0%), while students from previous cohorts averaged 12.7 (31.2%). The score distributions for both groups are displayed in Figure 3.

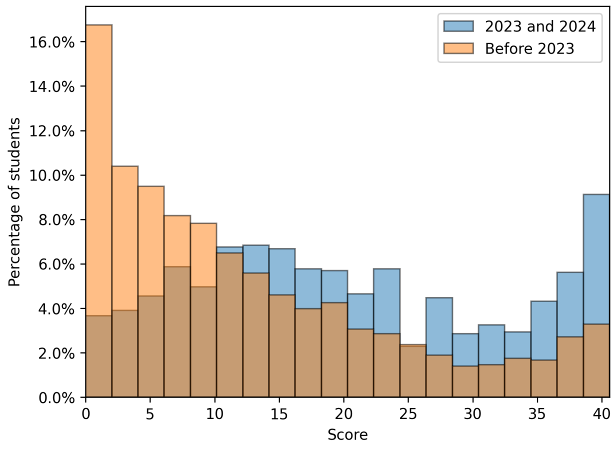

Fig. 3: Test score distribution of the 2023–2024 cohorts and those who graduated in prior years. Max score: 40.6.

4.3 Programming Tasks

The test consisted of 71 tasks across eight programming topics. For each task, students had the option to select "I don't know" if they were uncertain of the correct answer. Figure 4 illustrates the distribution of correct, incorrect, and "I don't know" responses for each topic. Statistical analysis revealed significant differences in the mean scores by graduation year. The results indicate that students from the 2023–2024 cohort consistently outperformed those from earlier cohorts across all programming topics.

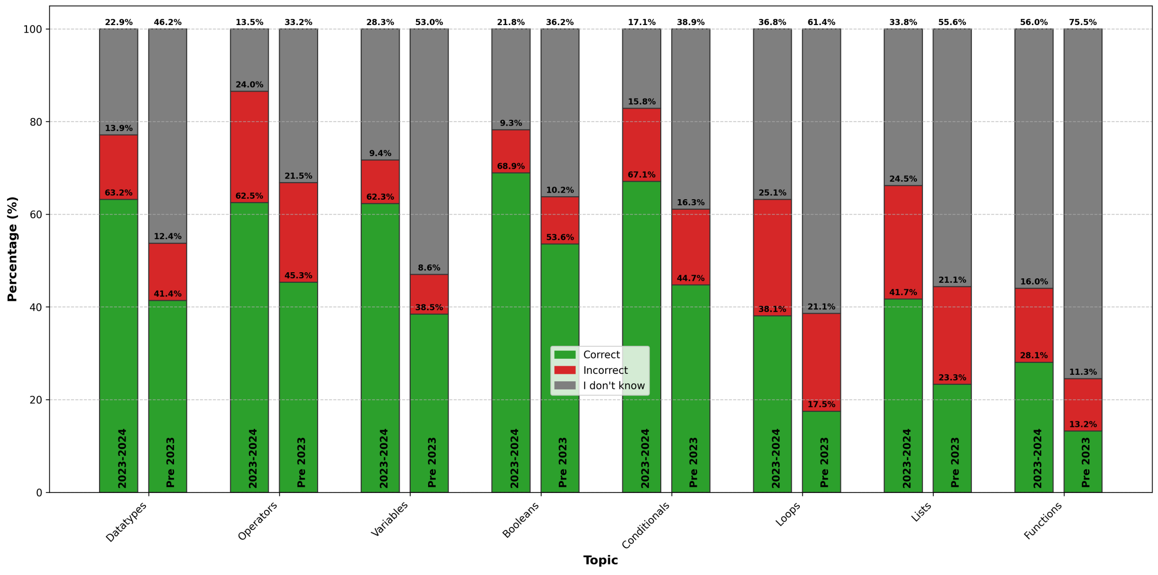

Fig. 4: Correct, incorrect and "I don't know" responses to each programming concept category.

5 Discussion

5.1 RQ1: The Impact of the Curricular Reform

The results clearly indicate that the curricular reform has positively impacted students' programming competency. The 2023–2024 cohort demonstrated a mean score improvement of 18.8%, performing significantly better on the test than previous cohorts. This improvement was consistent across all programming topics, where the 2023–2024 cohort outperformed earlier groups in every category.

However, a notable portion of students still performed at the lower end of the scale, with 18% of the 2023–2024 cohort scoring below 20%. While this could be attributed to teachers limited experience in teaching programming, likely affecting the students' learning outcomes, it is important to note that programming is widely recognized as a difficult subject to master [22, 24, 44]. Although the test primarily assesses basic CS1 concepts, using the lower cognitive processes, it is expected that a substantial number of students may under-perform, irrespective of instructional quality.

Despite these challenges, the reform shows great promise. The 2023 students were exposed to programming throughout their three years of secondary school, while the 2024 students benefited from additional exposure in their 10th grade curriculum. By 2032, all students will have had computational thinking embedded as learning objectives from second grade onwards. Furthermore, the quality of instruction is likely to improve as teachers become more proficient with programming after more years of exposure. We anticipate that the outcomes of this test will continue to improve in future years, ultimately fostering a society with increased digital competence.

5.2 RQ2: Is There Still a Need for the CS1 Course?

Given that over 50% of students entering higher education have not been exposed to programming in the new curricular model, the CS1 course remains essential for the majority before they can advance to more specialized subjects. Even among the 2023-2024 cohort, many students exhibit a lack of fundamental programming competencies, struggling with basic programming concepts.

However, a substantial portion of students perform exceptionally well, raising questions about their educational trajectory. Among the 2023–2024 cohort, 14.7% scored above 90% on the test, and 6% of students from earlier cohorts achieved similarly high scores. These high-performing students may benefit from an accelerated curriculum that allows them to move beyond certain CS1 topics.

In the fall semester of 2024, the Department of Informatics at the University of Bergen introduced a supplementary test based on CS1 final exam tasks from previous semesters. Students who passed this test were exempted from all but one mandatory assignment towards the end of the CS1 course, allowing them to focus on other courses and further their computer science knowledge while only completing a single assignment and passing the final exam in CS1. Despite this opportunity, participation was relatively low. Of the 541 students enrolled, only 55 opted to take the supplementary test. The course instructor reported that several students with substantial programming experience chose not to participate. Reasons for this included a preference to "learn the subject properly" rather than skipping the course. Among the 55 participants, 31 students either met the passing criterion of 70% or scored between 60% and 70%. Those in the latter range were granted passes after a review of their specific errors.

It is important to note that traditional CS1 course content and structure diverge somewhat from the programming experiences encountered in K-12 education. In mathematics and natural science in K-12 programming is primarily emphasized as a tool for mathematical and scientific problem-solving rather than focusing solely on programming concepts. Additionally, our assessment of programming competency is limited to a subset of topics covered in the CS1 curriculum and evaluates students' ability to interpret code rather than write it. Therefore, while many students may achieve high scores, this does not necessarily indicate a comprehensive understanding of the full scope of CS1. As a result, some students may prefer to take the course to ensure they grasp the material fully before progressing to more advanced topics.

Discussions are ongoing across Norway regarding the potential for students to bypass the conventional CS1 course. While many students still benefit from learning programming from the ground up, an increasing number are proficient enough to begin with more advanced topics. These discussions have coalesced around two main proposals: the introduction of a *CS1.5* course and the option of *skipping CS1* altogether.

A growing number of students are entering higher education with a foundational knowledge of basic programming, though they may not yet be prepared to move directly into CS2 [16]. To address this, there are proposals for a new course, *CS1.5*, which would cater to students who already understand the basics but need instruction in the more advanced CS1 topics. This would allow them to complete a shortened course before transitioning to CS2.

The second option being considered is to allow certain students to skip CS1 entirely. Although this would apply to a small minority, our interpretation is that there is a subset of students proficient enough to move beyond CS1 and focus their time on more advanced coursework. Some institutions, such as the University of Stavanger, have discussed the possibility of allowing students who have completed the elective programming courses IT1 and IT2 to bypass CS1, as these courses largely align with the learning objectives of CS1.

While the number of students ready for an accelerated educational pathway remains small, this is expected to increase in the coming years as a result of the ongoing curricular reform. Implementing such pathways now seems both timely and necessary, as these changes would require significant administrative adjustments, which could take considerable time.

6 Conclusion

The findings of the Nordic Prior Knowledge Test in Programming so far highlight the positive effects of the recent curricular reform on students' programming competencies. The 2023–2024 students exhibited substantial improvements over cohorts from previous years. These results demonstrate that early exposure to programming in the K-12 period can significantly improve student performance across a range of programming topics. As programming becomes more integrated into earlier stages of education, we anticipate further improvements, particularly as teachers gain more experience in programming instruction.

Despite average gains for those students that underwent the new curriculum, many still struggle with basic concepts. This points to the inherent difficulties of learning programming and the need for relentless attention to instructional quality. As the curricular reform is fully implemented by 2032, with students entering higher education with programming experience since the second grade, we expect these challenges to diminish, ultimately fostering a more digitally competent society.

Regarding CS1, the majority of students still require this foundational course, especially those who have not been exposed to the new programming curriculum. However, the high performance of some students suggests that more advanced pathways may be necessary for high-achievers. By offering alternatives, such as exemption tests or accelerated curricula, institutions can better meet the diverse needs of their student populations.

In conclusion, while the curricular reform has already shown promising results, there remains a need for continued refinement of both teaching methods and course structures to accommodate the full range of student abilities. With these developments, we can better equip future generations with the programming skills essential for success in a digital world.

7 Further Work

7.1 Exploring Deeper Insights from the 2024 Dataset

The 2024 iteration of the Nordic Prior Knowledge Test in Programming has yielded a rich and comprehensive dataset, offering numerous opportunities for further research. Future publications will address the unanswered research questions posed in Section 1.1 delving deeper into the students' programming comprehension.

7.2 Administering the Test in Future Years

The Nordic Prior Knowledge Test in Programming is set to be administered annually, enabling longitudinal tracking of programming knowledge across the Nordic region. So far the test has been distributed in Norway and Sweden. Given the educational similarities among Nordic countries, we plan to extend the test to Denmark, Finland, and Iceland in future iterations.

7.3 Development of a Validated Test

For future test iterations, we are working on developing a concept inventory (CI), a validated assessment tool widely used in STEM fields [1, 3, 12, 15, 17, 21, 25, 36, 42]. The CI development process typically involves four stages [13]:

1. Determine the fundamental concepts
2. Identifying misconceptions
3. Develop questions
4. Validation

As of fall 2024, the first step, *Determine the Fundamental Concepts*, has been completed, and we are currently progressing through *Identify Misconceptions* and *Develop Questions*.

The goal of this CI is to address pre-test limitations which computer science CIs have historically had. We aim to create a test that aligns with the level of knowledge students possess upon entering higher education, which aligns with any (mis)conceptions formed during K-12 instruction. The CI is planned to be beta-tested in the next iteration of the test, scheduled for August 2025.

Acknowledgements

We extend our gratitude to all the experts who contributed to the development and distribution of the Nordic Prior Knowledge Test in Programming. Your involvement made this project possible. We also thank Robert Egelund for his careful proofreading of the data analysis. A special acknowledgment goes to Fredrik Manne for his instrumental role in initiating this project. We look forward to more fruitful research endeavors in the future.

References

1. Anderson, D.L., Fisher, K.M., Norman, G.J.: Development and evaluation of the conceptual inventory of natural selection. Journal of Research in Science Teaching **39**(10), 952–978 (2002). https://doi.org/10.1002/tea.10053
2. Anderson, L.W., Krathwohl, D.R.: Taxonomy for Learning, Teaching, and Assessing, A: A Revision of Bloom's Taxonomy of Educational Objectives, Complete Edition. Pearson (2001)
3. Bardar, E.M., Prather, E.E., Brecher, K., Slater, T.F.: Development and validation of the light and spectroscopy concept inventory. Astronomy Education Review **5**(2), 103–113 (2007). https://doi.org/10.3847/AER2006020
4. Bell, T., Andreae, P., Robins, A.: A case study of the introduction of computer science in NZ schools. ACM Trans. Comput. Educ. **14**(2) (Jun 2014). https://doi.org/10.1145/2602485
5. Bloom, B.S., Englehart, M.D., Furst, E.J., Hill, W.H., Krathwohl, D.R., et al.: Taxonomy of educational objectives, handbook I: the cognitive domain. New York: David McKay Co (1956)
6. Boccani, S., Chioccariello, A., Earp, J.: The Nordic approach to introducing Computational Thinking and programming in compulsory education. Report prepared for the Nordic@BETT2018 Steering Group (2018), `https://doi.org/10.17471/54007`
7. Bornat, R., Dehnadi, S., Simon: Mental models, consistency and programming aptitude. In: Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78. pp. 53–61. ACE '08, Australian Computer Society, Inc., AUS (2008), `https://dl.acm.org/doi/10.5555/1379249.1379253`

8. Caspersen, M.E., Larsen, K.D., Bennedsen, J.: Mental models and programming aptitude. In: Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education. pp. 206–210. ITiCSE '07, Association for Computing Machinery (2007). https://doi.org/10.1145/1268784.1268845
9. Dehnadi, S., Bornat, R., et al.: The camel has two humps (working title). Middlesex University, UK [unpublished manuscript] (2006), https://web.archive.org/web/20081225125506/http://www.cs.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf
10. Department for Education: The national curriculum in England: framework document (2014), https://web.archive.org/web/20240104161617/https://assets.publishing.service.gov.uk/media/5a7db9e9e5274a5eaea65f58/Master_final_national_curriculum_28_Nov.pdf
11. Duran, R.S., Rybicki, J.M., Hellas, A., Suoranta, S.: Towards a common instrument for measuring prior programming knowledge. In: Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education. pp. 443–449. ITiCSE '19, Association for Computing Machinery (2019). https://doi.org/10.1145/3304221.3319755
12. Epstein, J.: Development and validation of the calculus concept inventory. In: Proceedings of the 9th International Conference on Mathematics Education in a Global Community. vol. 9, pp. 165–170 (2007)
13. Evans, D., Gray, G., Krause, S., Martin, J., Midkiff, C., Notaros, B., Pavelich, M., Rancour, D., Reed-Rhoads, T., Steif, P., Streveler, R., Wage, K.: Progress on concept inventory assessment tools. In: 33rd Annual Frontiers in Education, 2003. FIE 2003. vol. 1, pp. T4G–1 (2003). https://doi.org/10.1109/FIE.2003.1263392
14. Frantsen, T.: Å vere lærar i programmering utan å kunne programmere. Master's thesis, Oslo Metropolitan University (2019), https://hdl.handle.net/10642/7795
15. Herman, G.L., Loui, M.C., Zilles, C.: Creating the digital logic concept inventory. In: Proceedings of the 41st ACM Technical Symposium on Computer Science Education. pp. 102–106. SIGCSE '10, Association for Computing Machinery (2010). https://doi.org/10.1145/1734263.1734298
16. Hertz, M.: What do "CS1" and "CS2" mean? investigating differences in the early courses. In: Proceedings of the 41st ACM Technical Symposium on Computer Science Education. pp. 199–203. SIGCSE '10, Association for Computing Machinery (2010). https://doi.org/10.1145/1734263.1734335
17. Hestenes, D., Wells, M., Swackhamer, G.: Force concept inventory. The Physics Teacher **30**(3), 141–158 (1992). https://doi.org/10.1119/1.2343497
18. Hubwieser, P.: Computer science education in secondary schools – the introduction of a new compulsory subject. ACM Trans. Comput. Educ. **12**(4) (nov 2012). https://doi.org/10.1145/2382564.2382568
19. Julie, H., Bruno, D.: Towards the identification of profiles based on the understanding of programming concepts: the case of the variable. In: 2019 IEEE Frontiers in Education Conference (FIE). pp. 1–8 (2019). https://doi.org/10.1109/FIE43999.2019.9028697
20. Julie, H., Bruno, D.: Approach to develop a concept inventory informing teachers of novice programmers mental models. In: 2020 IEEE Frontiers in Education Conference (FIE). pp. 1–9 (2020). https://doi.org/10.1109/FIE44824.2020.9274045
21. Libarkin, J.C., Anderson, S.W.: Assessment of learning in entry-level geoscience courses: Results from the geoscience concept inventory. Journal of Geoscience Education **53**(4), 394–401 (2005). https://doi.org/10.5408/1089-9995-53.4.394
22. Lister, R., Adams, E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J.E., Sanders, K., Seppälä, O., Simon, B., Thomas, L.: A multi-national study of reading and tracing skills in novice programmers. In: Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education. pp. 119–150. ITiCSE-WGR '04, Association for Computing Machinery (2004). https://doi.org/10.1145/1044550.1041673
23. Luxton-Reilly, A., Becker, B.A., Cao, Y., McDermott, R., Mirolo, C., Mühling, A., Petersen, A., Sanders, K., Simon, Whalley, J.: Developing assessments to determine mastery of programming fundamentals. In: Proceedings of the 2017 ITiCSE Conference on Working Group Reports. pp. 47–69. ITiCSE-WGR '17, Association for Computing Machinery (2018). https://doi.org/10.1145/3174781.3174784
24. McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B.D., Laxer, C., Thomas, L., Utting, I., Wilusz, T.: A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. SIGCSE Bull. **33**(4), 125–180 (Dec 2001). https://doi.org/10.1145/572139.572181
25. Mulford, D.R., Robinson, W.R.: An inventory for alternate conceptions among first-semester general chemistry students. Journal of Chemical Education **79**(6), 739 (2002). https://doi.org/10.1021/ed079p739

26. Norwegian Directorate for Education and Training: Læreplan i informasjonsteknologi lk20. https://data.udir.no/kl06/v201906/laereplaner-lk20/INF01-02.pdf, accessed: 2022-12-12
27. Norwegian Directorate for Education and Training: Læreplan i matematikk 1.–10. trinn (MAT01-05), https://www.udir.no/lk20/mat01-05, accessed: 2022-12-12
28. Norwegian Directorate for Education and Training: Læreplan i matematikk fellesfag 2P (MAT0504), https://www.udir.no/lk20/mat05-04/, accessed: 2024-09-12
29. Norwegian Directorate for Education and Training: Læreplan i matematikk fellesfag vg1 praktisk (matematikk P) (MAT08-01), https://www.udir.no/lk20/mat08-01, accessed: 2022-12-12
30. Norwegian Directorate for Education and Training: Læreplan i matematikk fellesfag vg1 teoretisk (matematikk T) (MAT09-01), https://www.udir.no/lk20/mat09-01, accessed: 2022-12-12
31. Norwegian Directorate for Education and Training: Læreplan i matematikk for realfag (matematikk R) (MAT0302), https://www.udir.no/lk20/mat03-02/, accessed: 2024-08-01
32. Norwegian Directorate for Education and Training: Læreplan i matematikk for samfunnsfag (matematikk S) (MAT0402), https://www.udir.no/lk20/mat04-02/, accessed: 2024-09-12
33. Norwegian Directorate for Education and Training: Læreplan i programmering og modellering X (PRM0102). https://www.udir.no/lk20/prm01-02, accessed: 2023-09-26
34. Norwegian Directorate for Education and Training: Kunnskapsløftet 2020 – hvorfor har vi fått nye læreplaner? (2021), https://www.udir.no/laring-og-trivsel/lareplanverket/fagfornyelsen/hvorfor-nye-lareplaner, accessed: 2024-09-04
35. Norwegian Directorate for Education and Training: Eksamen REA3058 matematikk R2 (05 2023), https://sokeresultat.udir.no/eksamensoppgaver.html?query=REA3058&ExPeriodName=2024-1, accessed: 2024-09-23
36. Parker, M.C., Guzdial, M., Engleman, S.: Replication, validation, and use of a language independent CS1 knowledge assessment. In: Proceedings of the 2016 ACM Conference on International Computing Education Research. pp. 93–101. ICER '16, Association for Computing Machinery (2016). https://doi.org/10.1145/2960310.2960316
37. Porter, L., Garcia, S., Tseng, H.W., Zingaro, D.: Evaluating student understanding of core concepts in computer architecture. In: Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education. pp. 279–284. ITiCSE '13, Association for Computing Machinery (2013). https://doi.org/10.1145/2462476.2462490, https://doi.org/10.1145/2462476.2462490
38. Skolverket: Curriculum for the compulsory school, preschool class and school-age educare (2018), https://www.skolverket.se/download/18.31c292d516e7445866a218f/1576654682907/pdf3984.pdf, accessed: 2024-09-12
39. Stenlund, E.: Programmering og Fagfornyelsen. Master's thesis, University of Oslo (2021), http://urn.nb.no/URN:NBN:no-89753
40. Taylor, C., Zingaro, D., Porter, L., Webb, K.C., Lee, C.B., Clancy, M.: Computer science concept inventories: past and future. Computer Science Education **24**(4), 253–276 (2014). https://doi.org/10.1080/08993408.2014.970779
41. Tew, A.E.: Assessing fundamental introductory computing concept knowledge in a language independent manner. Ph.D. thesis, Georgia Institute of Technology (2010), http://hdl.handle.net/1853/37090
42. Tew, A.E., Guzdial, M.: The FCS1: a language independent assessment of CS1 knowledge. In: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education. pp. 111–116. SIGCSE '11, Association for Computing Machinery (2011). https://doi.org/10.1145/1953163.1953200
43. Thompson, E., Luxton-Reilly, A., Whalley, J.L., Hu, M., Robbins, P.: Bloom's taxonomy for cs assessment. In: Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78. pp. 155–161. ACE '08, Australian Computer Society, Inc., AUS (2008), https://dl.acm.org/doi/10.5555/1379249.1379265
44. Watson, C., Li, F.W.: Failure rates in introductory programming revisited. In: Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education. pp. 39–44. ITiCSE '14, Association for Computing Machinery (2014). https://doi.org/10.1145/2591708.2591749
45. Webb, K.C., Taylor, C.: Developing a pre- and post-course concept inventory to gauge operating systems learning. In: Proceedings of the 45th ACM Technical Symposium on Computer Science Education. pp. 103–108. SIGCSE '14, Association for Computing Machinery (2014). https://doi.org/10.1145/2538862.2538886, https://doi.org/10.1145/2538862.2538886
46. Xie, B., Davidson, M.J., Li, M., Ko, A.J.: An item response theory evaluation of a language-independent cs1 knowledge assessment. In: Proceedings of the 50th ACM Technical Symposium on Computer Science Education. pp. 699–705. SIGCSE '19, Association for Computing Machinery (2019). https://doi.org/10.1145/3287324.3287370, https://doi.org/10.1145/3287324.3287370