What are the Fundamentals of Norwegian CS1?

Sondre S. Bolland, Aleksandr Popov, Tyra F. Eide, and Torstein J. F. Strømme

University of Bergen

{sondre.bolland, aleksandr.popov, tyra.eide, torstein.stromme}@uib.no

Abstract. The introductory programming course, known as CS1, has evolved significantly since its inception, with diverse opinions on the essential concepts that should be included. This study aims to identify the fundamental concepts taught in Norwegian CS1 courses in order to develop a validated assessment tool: a concept inventory. This tool will be utilized in the Nordic Prior Knowledge Test in Programming, which is designed to assess the pre-existing programming knowledge of students entering higher education. This test uses Python, the dominant programming language in both K-12 and higher education in Norway.

To identify the fundamentals of CS1 we employed a triangulation approach that included three perspectives: the intended curriculum, the assessed curriculum, and the experienced curriculum. Our methodology involved a Delphi process with Norwegian CS1 educators, an analysis of final exams from various Norwegian institutions, and surveys of computer science students regarding the difficulty and importance of programming concepts.

Our findings reveal that concepts related to looping, functions, conditionals and error interpreting are central to Norwegian CS1 courses, aligning with existing literature. However, we also identified notable discrepancies compared to older CS1 concept studies developed in other countries, particularly in concepts like recursion, data structures beyond arrays/lists and maps, and test design. These results underscore both the dynamic nature of computer science education and the enduring importance of foundational topics that students are expected to master.

Keywords: Computer Science · CS1 · Introduction to Programming · Concept Identification · Concept Inventory

1 Introduction

The term *CS1* originates from the ACMs 1978 Computing Curricula, where it was designated as the label for the introductory programming course [6]. While the names and general principles of CS1 have remained consistent, the past 46 years have seen significant changes in the concepts covered in these courses. Hertz [16] found a lack of consensus among U.S. computer science educators regarding which concepts should be included in CS1 (and CS2[1]) and the relative importance of these concepts. Even when limiting the analysis to whether a concept was deemed essential or whether students were expected to demonstrate any mastery of it, Hertz found persistent disagreements among instructors.

Analogously, institutions in different countries and continents may also have divergent opinions on the content of CS1. We hypothesize that this phenomenon is present in Norway, potentially exacerbated by the recent expansion of CS1 courses at Norwegian institutions to include all STEM students, beyond just computer science students. This broader audience may dilute the focus and content of CS1 courses, as they now must appeal to geologists, chemists, biologists, and others. Consequently, the curriculum might vary significantly from one CS1 course to another.

Despite these variations, there may be a foundational core of concepts that all CS1 students must understand to succeed in their course. Mapping these concepts would be beneficial for researchers and educators to better develop teaching strategies and educational materials, including validated assessment.

[1] CS2 is the second computer science course, taken after CS1, typically focusing on data structures and algorithms.

This study aims to determine the most fundamental concepts of CS1, with the specific purpose of developing a concept inventory (CI). A CI serves as a criterion-referenced assessment tool designed to gauge students' precise understanding of predefined concepts. Comprising meticulously crafted multiple-choice questions, a CI exposes conceptual misconceptions and guides instructors in refining teaching methodologies and curricula [29].

The envisioned CI addresses a novel challenge in Norwegian computer science education: the integration of programming into the K-12 curriculum. With the latest curriculum update, "Kunnskapsløftet 2020," programming and algorithmic thinking have been incorporated into subjects such as mathematics, science, music, and arts and crafts [1]. To gauge students' prior knowledge as they enter higher education, the *Nordic Prior Knowledge Test in Programming* was developed [2]. This assessment tool evaluates students' proficiency in fundamental programming concepts through tasks implemented in Python, the language emphasized in the K-12 curriculum [28] and the dominating language in introductory higher education courses, as confirmed by our survey of CS1 educators. The test's purpose is to guide CS1 educators in allocating teaching resources effectively by identifying which concepts students are already familiar with and which require educational attention. Administered for the first time in 2023, the test is intended to be issued every fall semester to track the longitudinal progression of programming knowledge in the Nordics.

However, the programming tasks in the current test exhibit limitations. As of 2024, these tasks have been collaboratively developed by CS1 educators in Norway and Sweden, but without any validated methodology. Our objective is to create a CI to replace these programming tasks, offering a more standardized and reliable assessment method.

The proposed CI differs from existing CS1 CIs (such as the FCS1 [32] and the SCS1 [21]) by focusing on prior knowledge rather than knowledge developed throughout the CS1 learning period. We aim to create a test that assesses CS1 curriculum objectives, but with the level of knowledge developed during the K-12 period, before higher education instruction starts. Porter et al. [24], Tew and Guzdial [30], and Webb and Becker [34] suggest that student misconceptions in computer science likely originate from the instruction itself, rather than being pre-formed. As the content and structure of programming education in the K-12 period is different from instruction given at higher education, the misconceptions identified by those CIs may not align with the misconceptions held by incoming students. This discrepancy highlights the need for a new CI that accurately reflects the prior knowledge and potential misconceptions of incoming students.

1.1 Research Questions and Structure

This paper aims to address the first step in CI development [12]: *determining the fundamental concepts*. To identify the core elements of Norwegian CS1 curricula, we aim to answer the following research questions:

- **RQ1:** What concepts in Norwegian introductory programming courses are most *important* for students to learn?
- **RQ2:** What concepts in Norwegian introductory programming courses are most *difficult* for students to learn?

In this context, we define *importance* as the level of necessity to learn the concept before progressing to more advanced computer science curricula in later courses, and *difficulty* as the level of cognitive and practical challenge that students face when attempting to understand, apply, and master the concept.

In the following sections, we outline existing research on CS1 content and methods used to determine its core elements. We then describe our three-step approach to identifying the fundamental concepts of Norwegian CS1, followed by the results of these approaches. Finally, we provide an interpretation of these results and conclude what the key elements of Norwegian CS1 are.

2 Related Work

2.1 Historical and Recent Studies on Fundamental CS1 Concepts

Schulte and Bennedsen's 2006 study [26] provides a comprehensive survey of 349 CS1 educators (with 242 completed responses) aimed at identifying the key concepts considered essential in a CS1 course. The criteria for this identification included overall difficulty to learn, relevance to the subject, and the level of mastery based on Bloom's Taxonomy [7]. Their list of CS1 concepts, as compiled by Dale [9] and Milne [20], prominently features topics related to object-oriented programming (OOP). The survey revealed that 85% of the educators covered OOP in their courses, and 60% reported using an objects-first approach. Interestingly, the concept list also included topics emphasised in more advanced courses, such as *advanced data structures* (e.g., linked lists, trees), *algorithm design*, *algorithm efficiency* (Big-O notation), *recursion*, *generics*, and *divide and conquer* strategies. Notably, four of these advanced concepts ranked among the top five in terms of overall difficulty to learn. However, these concepts scored significantly lower in terms of perceived relevance. The concepts ranked most relevant were *Selection & Iteration*, *Simple Data structures*, *Parameters* and *Scope of Variables*.

Goldman et al. [13] conducted the first study on computer science CIs in 2008, identifying fundamental concepts in introductory programming. By surveying experts in the field, they identified and rated 32 concept categories by their importance and difficulty. 11 of these concepts were selected as the most fundamental based on an aggregate metric. These include:

1. Abstraction/Pattern recognition and use
2. Debugging, Exception Handling
3. Functional Decomposition, modularization
4. Conceptualizing problems, designing solutions
5. Designing tests
6. Inheritance
7. Memory model, references, pointers
8. Parameter scope, use in design
9. Procedure design
10. Recursion, tracing and designing
11. Issues of scope, local vs. global

The authors noted that reaching a consensus on the most important and difficult concepts for a CS1 course is inherently challenging due to the diversity of approaches in programming languages (e.g., Java, Python, C++), pedagogical paradigms (e.g., objects-first, objects-late, procedural), and programming environments used. As highlighted by Taylor et al. [29], computer science is a relatively young field where the content and underlying technology are significantly more fluid than in other fields. Consequently, the concepts most relevant to CS1 in 2006 and 2008 may differ from those relevant in 2024.

A more recent study by Luxton-Reilly et al. [19] in 2018 sought to establish which elements of syntax and semantics are covered by CS1 courses and what instructors expect from students. The study compiled concepts from nine prior sources [4, 11, 16, 18, 22, 25–27, 31], resulting in a comprehensive list of 12 overarching concepts, most encompassing various sub-concepts. These were used to analyze the CS1 learning outcomes from 103 courses at 101 universities across 12 countries (excluding Norway). The analysis aimed to determine which concepts are most frequently addressed in global CS1 courses. Table 1 presents these 12 overarching concepts and their respective frequencies of occurrence.

However, it is important to recognize that course learning objectives can often be vague and nonspecific. As Luxton-Reilly et al. [19] noted:

> *"The stated objectives for CS1 are often fairly abstract. In some cases, we concluded that they were too vague and generic to tag."*

This observation is echoed by practices at the Department of Informatics at the University of Bergen, where course learning objectives are intentionally written in a broad manner to allow

Concept	Number (%) of courses
Programming Process	90 (87%)
Abstract Programming Thinking	65 (63%)
Data Structures	41 (40%)
Object-Oriented Concepts	37 (36%)
Control Structures	34 (33%)
Operations and Functions	27 (26%)
Data Types	24 (23%)
Input/Output	18 (17%)
Libraries	15 (15%)
Variables and Assignment	14 (14%)
Recursion	10 (10%)
Pointers & Memory Management	5 (5%)

Table 1: Number of courses whose objectives address each broad concept from the master list (with percentage in parentheses).

for flexibility. Because these objectives can only be revised one year in advance, instructors tend to draft them in a way that allows for content adjustments each semester without the need for extensive pre-planning.

In Henry and Dumas's 2020 study [15], the development of a CI for programming fundamentals involved closely observing students during a 10,000-line coding project. The researchers identified important concepts with frequent misconceptions by observing students both individually and in groups, and by engaging in informal discussions with them throughout the semester. Weekly meetings with the four instructors in charge of the course, along with a review of exam results and consultations with other experts and educators, helped to refine the identification of problem areas. Their findings highlighted three key concepts–*conditionals*, *functions*, and *variables*–as essential for the CS1 course, which then became the focus of their CI development.

Norwegian CS1 Content. Research on CS1 content predominantly comes from outside Norway and even beyond the Nordic countries. Anecdotal comparisons between the content typically found in Norwegian CS1 curricula and the literature reveal notable differences in concept emphasis. For instance, concepts such as *recursion*, *memory management, references, pointers*, and *designing tests* are frequently absent in many Norwegian CS1 courses. Additionally, while international studies often emphasize object-oriented concepts, these are not a major focus in many Norwegian CS1 curricula. Our survey of Norwegian CS1 educators indicates a predominant use of procedural programming paradigms.

While these discrepancies may stem from the evolving nature of computer science and the age of the studies, they nevertheless underscore the need to closely examine the fundamental concepts in modern CS1 courses in Norway and how these align with those identified in the broader international context.

2.2 Consulting CS1 Experts

The sources mentioned in the previous section employ a diverse range of methods to determine which concepts in CS1 are considered fundamental for the course. However, the most prominent methodology for determining fundamental concepts in the development of computer science CIs is the *Delphi process*. A 2022 literature review by Ali et al. [3] identified 65 papers on computer science CIs. Of these, 22 discussed the process of selecting concepts, with 17 utilizing the Delphi process.

The Delphi process is a structured communication technique used to achieve consensus among experts in a particular field. It employs a four-step approach (described in Section 3.1) to gather and rank a collection of concepts within the selected field. These concepts are iteratively rated

with the goal of facilitating the sharing of opinions among the experts, ultimately aiming to reach a consensus on the topic at hand [10]. In the development of computer science CIs, it is common to define the fundamental concepts by their importance and difficulty to learn [3].

While the Delphi process is the favored method, we propose that it may have limitations when applied to CI development. Although experts are generally knowledgeable about difficult concepts, there can be a disconnect between the intended learning outcomes and the curriculum that students actually experience [17]. Experts generally know what is difficult for students but occasionally underestimate the difficulty of some topics [14]. Furthermore, self-report instruments are often criticized regarding the extent to which they are related to actual behavior [5].

To address these potential shortcomings, we have augmented the Delphi process with two additional approaches to obtain a more accurate representation of the curriculum as it is truly implemented.

3 Methodology

To define the scope of CS1 in Norway, we employed a triangulation approach using three distinct perspectives: the *intended curriculum*, the *assessed curriculum*, and the *experienced curriculum*.

The *intended curriculum* refers to the official, planned curriculum, encompassing the goals, objectives, content, and learning outcomes that educators intend for students to achieve [17]. The *assessed curriculum* focuses on the elements that are evaluated through formal assessments, such as assignments and exams, thus highlighting what is actually measured in terms of student learning and performance [23]. The *experienced curriculum*, also known as the operational curriculum, pertains to the curriculum as it is actually experienced by students in the classroom. This dimension is subjective, varying according to factors such as teaching methods, classroom dynamics, individual student backgrounds, and the broader learning environment [17].

To capture the *intended curriculum*, we consulted experts using the Delphi process. The *assessed curriculum* was examined through a mixed-methods analysis of CS1 exams administered across various institutions in Norway. The *experienced curriculum* was explored by gathering insights from students enrolled in computer science courses.

3.1 Intended Curriculum - Delphi Process

We engaged CS1 experts from universities and university colleges across Norway, including higher education professionals involved in teaching CS1 or conducting CS1-related research, to participate in a Delphi process. These experts took part in a series of online surveys designed to identify the most important and challenging concepts within the Norwegian CS1 curriculum.

While the recommended number of participants for a Delphi process is typically between 15 and 30 [8], we only secured the participation of nine experts, reflecting the relatively small pool of CS1 educators in Norway. By the final round of surveys, participation dropped to six, likely due to the study's timing during the exam period, which limited educators' availability. Despite these seemingly small numbers, the participating educators covers more than half of institutions affiliated with the *National Academic Council for ICT*[2] in Norway. Consequently, the views and perspectives gathered likely provide a representative overview of the landscape of Norwegian computer science higher education.

This Delphi process was modeled after the methodology outlined by Goldman et al. [13], who conducted a similar process for CS1 in the US. In their paper they describe a number of possible improvements on their work which we have employed in our methodology. The Delphi process consisted of four surveys:

1. Concept identification
2. Initial rating
3. Negotiation
4. Final rating

[2] Translated from Norwegian: *Nasjonalt fagorgan for IKT*.

Initially, each expert was asked to list 10 to 15 programming concepts from their CS1 curriculum, based on their importance and difficulty. Specifically, they were asked to name concepts which they considered crucial for students to learn to succeed in CS1, and those most challenging for students to grasp. These concepts were then categorized into general concept categories through inductive coding [33] conducted by the first two authors.

Subsequently, the same experts rated these coded concept categories in a second survey using the same two criteria. Each concept category was rated on a scale from 1 to 10, where 1 indicated the least importance/difficulty and 10 indicated the greatest importance/difficulty. An eleventh option, "Not Relevant for CS1," was available for concepts deemed unsuitable for the CS1 curriculum.

In the third survey, participants were presented with the same items, but with statistical data about the results from the previous survey: the mean, standard deviation and the distribution in form histograms. If a participant's rating in the third survey fell outside the interquartile range (middle 50%) of the ratings from the second survey, they were prompted to provide a justification for their rating.

The fourth survey presented the same items again, along with the statistics and justifications from the third survey, resulting in a final rating.

3.2 Assessed Curriculum - Exam analysis

The second approach involved analyzing the most recent exams from different institutions in Norway. Final exams are designed to assess the extent to which students have mastered the course objectives, and thus, they should include the concepts that lecturers deem most important. To determine whether the concepts rated as important align with those assessed in exams, we requested CS1 instructors in Norway to send us the exam tasks from a recent semester. We then used the list of concept categories developed from the first Delphi survey to identify which concepts were needed to solve these tasks. We used two metrics:

- Number of concept category occurrences
- Number of occurrences of what the authors interpreted as the main concept category(ies) being assessed

As an example, one of the exam tasks asked the student to determine the output of this code snippet:

```
t1 = 6
t2 = 0
resultat = 0
while resultat <= 9:
    if t1 < t2:
        resultat += t1
    else:
        resultat += t2
    t1 -= 1
    t2 += 2
print(resultat)
```

For this task, we identified the occurrence of the concept categories *Conditional Execution*, *Looping*, *Primitive Datatypes - Mathematical Operators*, *Primitive Datatypes - Declaration*, *Boolean Expressions*, *Variable Semantics*, and *Sequential Execution*. We interpreted the main concepts being assessed as both *Looping* and *Conditional Execution*.

For each exam task, we counted these two metrics and weighed them by the number of points the tasks allocated.

The analysis was conducted by the first three authors. Initially, all three authors independently analyzed the same exam and discussed which concept categories were present and which were intended as the primary focus of assessment until a consensus was reached. After this calibration, the remaining exams were divided among the authors for individual analysis.

3.3 Experienced Curriculum - Student Surveys

The third approach involved surveying the students about their experiences with the CS1 curriculum. While lecturers may possess a general understanding of which concepts are difficult for students to learn, there could be a misalignment between the perspectives of students and lecturers.

To gather the student's perspective, we reused the second survey from the Delphi process, asking students to rate each concept category based on importance and difficulty. This survey was administered to two student groups: those who had recently completed their CS1 course and those who had just finished their CS2 course, both at the Department of Informatics at the University of Bergen. Surveying both these groups aimed to capture perspectives from those who had recently acquired the material and from more experienced students who had the opportunity to apply CS1 concepts in a CS2 context. The idea was that one might consider a concept more or less difficult/important after having utilized the concept in a larger and more advanced context in CS2.

4 Results

4.1 Delphi Process

In the first survey we collected a total of 94 concepts from the nine experts. Some of the given concepts were excluded from the dataset due to the authors interpreting them as too broad or vague for any one concept category. This included the concepts "*translate mathematical, scientific models to code*" and "*identifying how to interpret tasks and how to solve*." The concepts where coded into 20 concepts categories. The categories and their description can be found in Appendix A.

These 20 concept categories were evaluated iteratively over three surveys. Figure 1 presents the final importance and difficulty ratings as a scatter plot. Each data point includes error bars representing the standard deviation of each metric along their respective axes. Detailed mean values and standard deviations are provided in Appendix A.

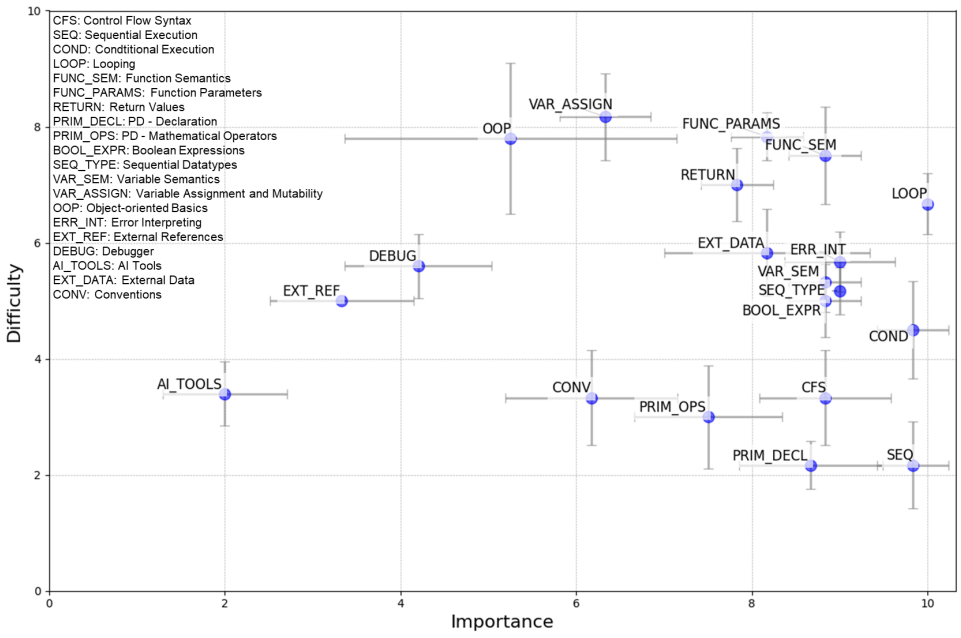

Fig. 1: Importance and difficulty ratings from the final survey (n=6). Each concept category has error bars representing the standard deviation for the two metrics along the respective axes.

4.2 Exam analysis

We collected six exams from various Norwegian institutions, each representing distinct CS1 courses. Our analysis revealed that certain concept categories were present in nearly every exam task. These categories included:

- *Primitive Datatypes - Declaration* (all subtasks)
- *Variable Semantics* (all subtasks)
- *Sequential Execution* (145 out of the total 150 subtasks)

Given that the use of primitive data types, variables, and sequential execution are fundamental to almost all programming, their presence in the exam tasks does not provide us with much information. Therefore, these categories have been excluded from the subsequent results and analysis.

Each exam allocated 100 points, summing up to a total of 600 points across all exams. Table 2 presents the occurrence percentages of various concept categories and the frequency with which each category was identified as the main focus of assessment in the tasks. This is calculated as the number of points each concept category received divided by the total points for all exams.

Concept	**Occurrences**	**Main Concepts**
Control Flow Syntax	58%	3%
Condtitional Execution	50%	30%
Looping	62%	22%
Function Semantics	64%	19%
Function Parameters and Arguments	65%	3%
Return Values	61%	1%
Primitive Datatypes - Mathematical Operators	47%	3%
Boolean Expressions	49%	2%
Sequential Datatypes	70%	34%
Variable Assignment and Mutability	16%	3%
Object-oriented Basics	24%	21%
Error Interpreting	5%	0%
External References	12%	0%
Debugger	2%	2%
AI Tools	0%	0%
External Data	16%	15%
Conventions	11%	6%

Table 2: Concept occurrences in exam tasks multiplied by the number of points the task allocated, divided by the total number of points for all exams.

To analyze the variation in focus on different concept categories across CS1 exams, we calculated the mean and standard deviation of the occurrences for each category between the exams (see table in Appendix B). The results show significant variability, with many concept categories displaying high standard deviations, some even exceeding their respective means. This indicates substantial inconsistency in how these concepts are assessed. Notable categories with a standard deviation larger than their mean include *Conventions*, *Debugger*, *External References*, *Error Interpreting*, *Object-Oriented Basics*, and *Variable Assignment and Mutability*.

In contrast, the three categories with the highest consistency–indicated by the smallest relative difference between mean and standard deviation–were *Conditional Execution*, *Looping*, and *Sequential Datatypes*, making them the most consistently assessed topics across exams. Interestingly, *AI Tools* was not assessed even once.

4.3 Student Perceptions of Difficulty and Importance

The concept categories were rated by 55 CS1 and 88 CS2 students. Note that the concept category *Object-oriented Basics* was excluded from these surveys because it was not covered in their respective CS1 courses, thereby preventing these students from evaluating its difficulty and importance.

Due to the non-normality of the distribution of these ratings, we utilized a non-parametric test (Mann-Whitney U test) to assess differences in ratings between CS1 and CS2 students. Among the 19 concept categories evaluated, only three importance ratings and three difficulty ratings showed statistically significant differences in means. The specific rating values of these student groups can be found in Appendix C.

Given the similarity in their ratings, we combined both groups into a single total student rating (n=143). Table 3 presents a comparison between the ratings of this combined student group and those of the experts. Using the same non-parametric test, we found four importance ratings and five difficulty ratings to be significantly different in their means, as indicated in the table.

	Importance		**Difficulty**	
Concept Category	**Students**	**Experts**	**Students**	**Experts**
Control Flow Syntax	8.70	8.83	4.74	3.33
Sequential Execution†‡	8.35	9.83	4.81	2.17
Conditional Execution	8.60	9.83	4.06	4.50
Looping	8.66	10.00	5.38	6.67
Function Semantics	8.45	8.83	5.51	7.50
Function Parameters and Arguments‡	7.98	8.17	4.75	7.83
Return Values‡	8.13	7.83	4.42	7.00
Primitive Datatypes - Declaration	7.83	8.67	3.48	2.17
Primitive Datatypes - Mathematical Operators	7.83	7.50	3.64	3.00
Boolean Expressions	8.22	8.83	3.92	5.00
Sequential Datatypes	7.73	9.00	5.22	5.17
Variable Semantics‡	7.91	8.83	3.84	5.33
Variable Assignment and Mutability‡	7.38	6.33	5.27	8.17
Error Interpreting	8.65	9.00	6.45	5.67
External References†	7.37	3.33	5.02	5.00
Debugger†	7.09	4.20	5.55	5.60
AI Tools†	6.23	2.00	3.54	3.40
External Data	6.61	8.17	6.50	5.83
Conventions	6.94	6.17	4.79	3.33

Table 3: The mean importance and difficulty ratings by all students (n=143) and experts (n=6). "†" denotes significant difference of importance mean. "‡" denotes significant differences in difficulty mean.

5 Discussion

5.1 Course Variation in Norway

Our findings reveal disparities in how experts perceive the importance and difficulty of various concept categories in Norwegian CS1 courses. In the initial rating survey, 16 out of 20 topics showed a standard deviation above 1.00 for importance ratings. The most contentious categories–those with a standard deviation exceeding 2.00–included *Primitive Datatypes - Declaration*, *Object-oriented Basics*, *Debugger*, *AI Tools*, and *Conventions*.

Several of these contentious categories are notable for their variability. For example, *AI Tools* is an emerging technology, with applications like OpenAI's ChatGPT[3] having been introduced only

[3] https://openai.com/chatgpt/

recently. Its inclusion in curricula has been debated, particularly concerning issues of plagiarism, which may explain its limited presence in established coursework. *Conventions* is often seen as a concept that is implicitly learned throughout various courses, rather than explicitly taught. Object-oriented programming topics, on the other hand, seem to be relegated to more advanced courses, with two experts deeming them "Not relevant for CS1" in their final ratings. The majority of experts indicated that they teach CS1 using a procedural paradigm.

Our analysis of final exams across different CS1 courses further highlights this variability. Many concept categories exhibited a standard deviation greater than the mean. *Object-oriented Basics* showed particularly high variability: out of the six exams analyzed, only two placed major emphasis on this topic. The remaining four either excluded it entirely or allocated at most 5% of the total points to it.

Despite the variation across many topics, some concept categories exhibited consistently low variability between exams, including *Conditional Execution*, *Looping*, and *Sequential Datatypes*. These categories were also rated highly by experts, with importance ratings of nine or higher, indicating strong alignment between expert priorities and the content emphasized in exams.

It is important to note that the exam analysis offers only a partial perspective on the concepts assessed in these courses. In post-analysis discussions, several experts noted that practical skills, such as error interpretation and resolution, were primarily evaluated through assignments, leading to their diminished presence in final exams.

The students' ratings generally aligned with those of the experts, with 15 out of 19 concept categories (excluding *Object-oriented Basics*) showing no significant difference in perceived importance or difficulty. Interestingly, in cases where significant differences in difficulty ratings were observed, students consistently rated the concepts as easier than the experts did. These same categories were also rated highly in importance by the experts. A possible explanation for this discrepancy is that instructors, recognizing these topics as both challenging and crucial, may devote extra attention to them in their teaching. This could lead students to perceive these topics as easier simply because they have been taught thoroughly.

5.2 Aggregating Importance and Difficulty for Concept Ranking

Despite the initial variability in the expert ratings, a notable convergence in opinions emerged by the final round of ratings, as evidenced by a reduction in standard deviation across all concept categories. By the end, only *Object-oriented Basics* and *External Data* retained a standard deviation above 1.00 for both importance and difficulty, suggesting growing consensus among the experts on most topics.

To decide which concept categories were most fundamental to the CS1 course we computed a single aggregate metric to rank the topics. Given that the primary purpose of these fundamental concepts is to define the scope for assessing the knowledge of incoming students, our focus is on the more basic elements of CS1. As a result, we prioritize the importance of the topics over their difficulty. To reflect this prioritization, we adjusted the weighting of difficulty by applying a square root transformation, resulting in the following formula:

$$Aggregate = Importance \cdot \sqrt{Difficulty}$$

This method yielded the top-ranking concept categories shown in Table 4 (the complete list can be found in Appendix A). These six concept categories featured frequently on final exams, with the notable exception of *Error Interpreting*. The importance ratings for these categories showed no significant difference between students and experts. For difficulty, students generally rated these topics as either equally challenging or easier compared to the experts.

5.3 How do the Findings Compare to the Literature?

Common Concepts. Among the top-ranked concept categories identified through our aggregate metric, several align closely with findings from previous research, underscoring their significance in introductory programming courses.

Concept	Expert Importance	Student Importance	Expert Difficulty	Student Difficulty	Exam Task Occurrence	Exam Task Main Concept
1. Looping	10	8.66	6.67	5.38	62%	22%
2. Function Semantics	8.83	8.45	7.5	5.51	64%	19%
3. Function Parameters	8.17	7.98	7.83	4.75	65%	3%
4. Error Interpreting	9	8.65	5.67	6.45	5%	0%
5. Condtitional Execution	9.83	8.6	4.5	4.06	50%	30%
6. Return Values	7.83	8.13	7	4.42	61%	1%

Table 4: Top six concept categories as determined by the aggregate metric, including importance and difficulty ratings by experts and students, and exam occurrences.

Looping emerged as our highest-ranked category, with ratings from our experts closely matching those in Goldman et al.'s study [14], where it received an importance score of 9.5 and a difficulty score of 6.6. This alignment is further supported by Schulte and Bennedsen [26], who emphasized the significance of *Selection and Iteration*, identifying it as the most relevant concept for CS1 courses.

Function-related concepts have been repeatedly highlighted as crucial within CS1 curricula. Henry and Dumas [15] identified functions, along with conditionals and variables, as key concepts during the dropout peak in introductory programming courses. Their mixed-methods approach targeted functions as one of three core topics for their CI, noting that students found it particularly challenging to articulate misconceptions about functions, indicating the complexity of this concept. Similarly, Goldman et al. [14] included five function-related concept categories, which yielded an average importance score of 8.8 and a difficulty score of 7.5. Schulte and Bennedsen [26] also highlighted the relevance of understanding function *Parameters*, particularly in terms of the correct type and number for a given function or method, ranking it as the third most relevant concept, though with a low difficulty score (2.8 out of 5.0).

Conditional Execution is another key concept recognized in multiple studies, though generally not considered highly difficult. Henry and Dumas [15] included this topic in their CI, observing that it was the easiest of the three key concepts they identified. Goldman et al. [14] rated *Conditionals* with a 9.3 in importance and 6.6 in difficulty, indicating it is more challenging than our findings or those of Henry and Dumas suggest. Interestingly, Schulte and Bennedsen [26] did not include a concept related to conditionals in their study.

Error Interpretation, as defined in our study, is somewhat narrower than how it has been categorized in other research. Schulte and Bennedsen [26] use the broader term *Debugging*, while our focus is on the identification of errors rather than the additional process of resolving them. They rated the topic 3.8 out of 5 on relevance and 3.2 on overall difficulty. Goldman et al.'s [14] category *Debugging/Exception Handling*: *"Developing and using practices for finding code errors."* aligns more closely with ours. Their expert panel rated this topic identically to ours on importance (9.0 out of 10.0) but considered it significantly more difficult, with a score of 8.6 compared to our 5.6. Luxton-Reilly et al. [19] included *Debugging* within their broader category *Programming Process*, which was identified as a topic covered by 87% of the CS1 courses in their study.

Diverging Concepts. Among the concept categories that were rated and analyzed in our study, some diverge from the existing literature. One of the most notable shifts has been in the emphasis on object-oriented programming (OOP). Schulte and Bennedsen [26] reported that in 2006, 85% of educators included OOP in their CS1 courses, with 60% adopting an objects-first approach. However, in our analysis, OOP was scarcely featured in Norwegian CS1 assessments apart from two courses and was even deemed "Not relevant for CS1" by two of the six CS1 experts in the final survey.

Additionally, other concepts frequently highlighted in past studies were absent from those mentioned by Norwegian educators. These concepts, along with the corresponding studies, include:

– *Recursion* [14, 16, 18, 22, 25–27, 31]

- *Data structures* such as matrices [18], graphs [16] and collections beyond arrays [25, 27]
- *Test design* [14, 16, 27]

Whether these curricular differences are due to geographical or temporal factors remains uncertain. However, this divergence underscores the fluidity and evolving nature of computer science education, where priorities and approaches can shift significantly over time and across regions.

5.4 Limitations

The recommended number of experts for a Delphi process ranges from 15 to 30 [8]. However, we initially secured only nine experts, and this number further decreased to six by the final survey. In the first survey, each expert was asked to submit 10 to 15 concepts, resulting in 94 concepts, which were consolidated into 20 concept categories. By contrast, Goldman et al. [14] identified 32 concept categories. This discrepancy could suggest that our smaller expert pool may have led to fewer programming concepts being considered. However, it could also indicate that the concepts taught at Norwegian institutions are less varied than those in Goldman et al.'s study.

Comparing these concept categories, we find that ours are broader in scope. For example, Goldman et al. [14] delineated five distinct categories related to functions, whereas we identified only three. The broader nature of our categories may have influenced how experts assessed the importance and difficulty of these concept categories, potentially leading them to consider a wider range of differentiating aspects.

Additionally, the data used for the student perspective was collected from a narrow sample: the students of the Department of Informatics at the University of Bergen. Ideally, these students would have been sampled from multiple institutions in Norway, not only one. This may have skewed the student opinion to align with that of the authors' institution and not the more general opinion of computer science students across Norway.

In assessing the inclusion of concepts in final exams across institutions, a potential source of error lies in the second metric: *Number of occurrences of what the authors interpreted as the main concept category(ies) being assessed.* This analysis relied on educated guesses by the authors as to the course instructors' intended focus. In some cases, these interpretations may have been inaccurate, leading to potential misjudgments regarding the emphasis of certain concepts. Furthermore, the limited number of exams analyzed from distinct CS1 courses restricts the generalizability of our findings. A larger sample of exams would provide a more comprehensive and reliable assessment of the concepts emphasized. Additionally, extending the analysis to include assignments throughout the courses would enhance our understanding by offering a fuller view of the concepts assessed across the entire curriculum, not just in final exams.

6 Conclusion

This study has gathered insights into the most important and difficult concepts in the Norwegian CS1 curricula through expert consultation, assessment analysis, and student surveying. Initially, a notable variation in opinions emerged regarding the most relevant topics for the course. However, we have consolidated a few distinct topics that are rated highly by both students and experts and have been featured in many exam tasks across the nation. These topics, deemed highly relevant by existing literature spanning nearly two decades, include *Looping*, *Functions*, *Conditionals* and *Error Interpreting*.

Although many of the concept categories identified as both important and difficult are frequently assessed, *Error Interpreting* stands out due to its limited inclusion in exams, appearing in only 5% of all tasks. This limited inclusion may be due to its more frequent assessment in course assignments, or it may reflect the inherent challenge of designing efficient exam tasks to evaluate this skill. Therefore, there is a strong incentive to explore new methods for creating assessment tasks that effectively evaluate *Error Interpreting*, ensuring that this fundamental aspect of CS1 is adequately covered in the curriculum.

These selected topics require significant educational attention and are the concept categories we intend to use in developing a CS1 concept inventory. However, they are not the only ones in need of focus. Loops, conditionals and functions are all structures that depend on other programming concepts. For instance, in Python, a *for* loop cannot be written without using sequential datatypes, and a *while* loop and an *if-statement* cannot be written without using boolean expressions. This dependence is even more apparent when considering functions. A function modularizes code, implying that there is already some code with its set of concepts in the body of the function.

To accurately assess whether a student has understood looping, conditionals and function-related concepts, we must first ensure that they have understood their atomic parts. Therefore, in our continued work, we will also focus on the foundational components of our selected concept categories.

6.1 Further Work

The next phase in the CI development focuses on identifying misconceptions related to the selected concepts. Our objective is to analyze student perceptions acquired through both K-12 programming curricula and subsequently during the CS1 course. This approach aims to offer valuable insights into the effectiveness of K-12 programming instruction and to compare misconceptions that emerge before higher education with those developed during the initial stages of higher education.

Acknowledgements

We would like to thank all participating experts in the Delphi process, without whom this study and the entirety of The Nordic Prior Knowledge Test in Programming would not have been possible. Thank you to Siri Annethe Moe Jensen, Andreas Haraldsrud, Guttorm Sindre, Erlend Tøssebro, Nils-Christian Walthinsen Rabben and others for their participation. We look forward to more fruitful research endeavors in the future.

References

1. Kunnskapsløftet 2020. https://www.regjeringen.no/no/dokumentarkiv/regjeringen-solberg/aktuelt-regjeringen-solberg/kd/pressemeldinger/2018/fornyer-innholdet-i-skolen/id2606028/?expand=factbox2606064, retrieved: 2024-07-13
2. Nordisk forkunnskapstest i programmering. https://programmeringstesten.no/, retrieved: 2024-07-13
3. Ali, M., Ghosh, S., Rao, P., Dhegaskar, R., Jawort, S., Medler, A., Shi, M., Dasgupta, S.: Taking stock of concept inventories in computing education: A systematic literature review. In: Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 1. pp. 397–415 (2023)
4. Armstrong, D.J.: The quarks of object-oriented development. Communications of the ACM **49**(2), 123–128 (2006)
5. Artelt, C.: Lernstrategien in der schule. Handbuch Lernstrategien pp. 337–351 (2006)
6. Austing, R.H., Barnes, B.H., Bonnette, D.T., Engel, G.L., Stokes, G.: Curriculum'78: recommendations for the undergraduate program in computer sciencea report of the acm curriculum committee on computer science. Communications of the ACM **22**(3), 147–166 (1979)
7. 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)
8. Clayton, M.J.: Delphi: a technique to harness expert opinion for critical decision-making tasks in education. Educational psychology **17**(4), 373–386 (1997)
9. Dale, N.: Content and emphasis in CS1. ACM SIGCSE Bulletin **37**(4), 69–73 (2005)
10. Dalkey, N., Helmer, O.: An experimental application of the delphi method to the use of experts. Management science **9**(3), 458–467 (1963)
11. Draft, S.: Computer science curricula 2013. ACM and IEEE Computer Society, Incorporated: New York, NY, USA (2013)
12. Evans, D., Gray, G.L., Krause, S., Martin, J., Midkiff, C., Notaros, B.M., Pavelich, M., Rancour, D., Reed-Rhoads, T., Steif, P., et al.: Progress on concept inventory assessment tools. In: 33rd Annual Frontiers in Education, 2003. FIE 2003. vol. 1, pp. T4G–1. IEEE (2003)

13. Goldman, K., Gross, P., Heeren, C., Herman, G., Kaczmarczyk, L., Loui, M.C., Zilles, C.: Identifying important and difficult concepts in introductory computing courses using a delphi process. In: Proceedings of the 39th SIGCSE technical symposium on Computer science education. pp. 256–260 (2008)
14. Goldman, K., Gross, P., Heeren, C., Herman, G.L., Kaczmarczyk, L., Loui, M.C., Zilles, C.: Setting the scope of concept inventories for introductory computing subjects. ACM Transactions on Computing Education (TOCE) **10**(2), 1–29 (2010)
15. Henry, J., Dumas, B.: Approach to develop a concept inventory informing teachers of novice programmers mental models. In: 2020 IEEE frontiers in education conference (FIE). pp. 1–9. IEEE (2020)
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. p. 199203. SIGCSE '10, Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1734263.1734335, https://doi.org/10.1145/1734263.1734335
17. Hume, A., Coll, R.: Authentic student inquiry: The mismatch between the intended curriculum and the student-experienced curriculum. Research in Science & Technological Education **28**(1), 43–62 (2010)
18. Kelemen, C.F., Consortium, L.A.C.S., et al.: A 2007 model curriculum for a liberal arts degree in computer science. Journal On Educational Resources In Computing **7**(2) (2007)
19. 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 (2018)
20. Milne, I., Rowe, G.: Difficulties in learning and teaching programmingviews of students and tutors. Education and Information technologies **7**, 55–66 (2002)
21. 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 (2016)
22. Pedroni, M., Meyer, B.: Object-oriented modeling of object-oriented concepts: A case study in structuring an educational domain. In: International Conference on Informatics in Secondary Schools-Evolution and Perspectives. pp. 155–169. Springer (2010)
23. Porter, A.C., Smithson, J.L.: Alignment of assessments, standards and instruction using curriculum indicator data. In: Annual meeting of the American Educational Research Association, New Orleans (2002)
24. 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 (2013)
25. Sanders, K., Ahmadzadeh, M., Clear, T., Edwards, S.H., Goldweber, M., Johnson, C., Lister, R., McCartney, R., Patitsas, E., Spacco, J.: The canterbury questionbank: Building a repository of multiple-choice CS1 and CS2 questions. In: Proceedings of the ITiCSE working group reports conference on Innovation and technology in computer science education-working group reports. pp. 33–52 (2013)
26. Schulte, C., Bennedsen, J.: What do teachers teach in introductory programming? In: Proceedings of the second international workshop on Computing education research. pp. 17–28 (2006)
27. Simon, Chinn, D., de Raadt, M., Philpott, A., Sheard, J., Laakso, M.J., D'Souza, D., Skene, J., Carbone, A., Clear, T., et al.: Introductory programming: examining the exams. In: Proceedings of the Fourteenth Australasian Computing Education Conference-Volume 123. pp. 61–70 (2012)
28. Stenlund, E.: Programmering og Fagfornyelsen. Master's thesis (2021)
29. 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)
30. Tew, A.E.: Assessing fundamental introductory computing concept knowledge in a language independent manner. Georgia Institute of Technology (2010)
31. Tew, A.E., Guzdial, M.: Developing a validated assessment of fundamental CS1 concepts. In: Proceedings of the 41st ACM technical symposium on Computer science education. pp. 97–101 (2010)
32. 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 (2011)
33. Thomas, D.R.: A general inductive approach for qualitative data analysis (2003)
34. 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 (2014)

Appendix A

Delphi Concept Categories

Appendix A highlights the finer details of the Delphi process.

1 Expert Demographic

The initial survey included questions about the educational background of the participating experts. Out of the total nine respondents, eight were currently teaching or had previously taught CS1. The ninth expert was scheduled to teach CS1 in the upcoming semester and had both conducted CS1 research and authored educational materials for the subject.

All nine experts indicated that the programming language they had taught, were teaching, or would teach was Python. Additionally, one expert mentioned teaching JavaScript and Blockly.

The dominant programming paradigm taught was *procedural*, with six experts reporting this focus. One expert stated *object-oriented programming*, while two left the field blank or selected N/A.

Regarding research experience, four experts had conducted research specifically on CS1, four had advised PhD/Master/Bachelor students focusing on CS1-related topics, and two were inexperienced with CS1 research.

2 Coded Concept Categories

The coded concept categories and their descriptions are shown in Table 5.

3 Concept Category Ratings

The following two tables show the importance (see Table 6) and difficulty (see Table 7) ratings from the three rating surveys from the Delphi process. These tables include the mean and standard deviation of each step, in addition to the difference between the first and third rating.

From the importance and difficulty ratings, we computed a single metric to rank the topics. This metric was the product of the mean importance and the square root of the mean difficulty ratings:

$$Aggregate = Importance \cdot \sqrt{Difficulty}$$

Table 8 displays the concept categories ranked by this aggregate metric.

Concept Category	**Description**
Control Flow Syntax	Knowing how to syntactically write if-statements, loops and functions
Sequential Execution	Knowing how statements are executed one after another in the order they appear in the code
Conditional Execution	Knowing how to use conditional statements, such as if-else statements, to allow the program to make decisions based on certain conditions
Looping	Knowing how to use looping constructs, such as for loops and while loops, to allow the program to execute a block of code repeatedly until a certain condition is met
Function Semantics	Understanding when and how to use functions, to decompose and reuse code
Function Parameters and Arguments	Understanding function parameters (inputs defined within the function declaration) and arguments (actual values passed to the function)
Return Values	Understanding how functions can return data back to the caller, including use of None/Null
Primitive Datatypes -Declaration	Knowing of and how to create variables of primitive datatypes such as integer, float and boolean
Primitive Datatypes -Mathematical Operators	Knowing of and how to use mathematical operators on primitive datatypes such as addition, subtraction, division, multiplication and exponents, and the operator's precedence
Boolean Expressions	Being able to construct boolean expressions by use of equality, inequality, conjunction, disjunction, negation, greater than and less than
Sequential Datatypes	Knowing of and how to create variables of sequential datatypes, such as list and string, and manipulating these types by adding, replacing and deleting elements in the sequence
Variable Semantics	Knowing how to differentiate between variable and value and that variables references values, not expressions.
Variable Assignment and Mutability	Assigning a value to a variable by direct value or by another existing variable, and how this differentiates when using mutable and immutable datatypes
Object-oriented Basics	Knowing what a class and object is, along with constructor, instance methods and instance variables
Error Interpreting	Reading an error message, being able to identify the error and where the error is based on the error stack
External References	Aiding the coding process by use of resources such as official documentation, Google and Stack Overflow
Debugger	Aiding the coding process by use of an IDE debugger
AI Tools	Aiding the coding process by use of AI tools such as ChatGPT and Github Copilot
External Data	Reading and writing from/to external files such as CSV files
Conventions	Following coding standards such as naming conventions, formatting rules and documentation practices

Table 5: Label and description of concept categories.

Importance Ratings	1st		2nd		3rd		Difference	
Concept Category	**Mean**	**SD**	**Mean**	**SD**	**Mean**	**SD**	**Mean**	**SD**
Control Flow Syntax	8.29	1.80	8.50	1.38	8.83	0.75	-0.55	1.05
Sequential Execution	9.14	1.07	9.67	0.52	9.83	0.41	-0.69	0.66
Conditional Execution	9.43	0.79	9.67	0.52	9.83	0.41	-0.40	0.38
Looping	9.29	0.76	10.00	0.00	10.00	0.00	-0.71	0.76
Function Semantics	8.29	1.11	8.17	1.60	8.83	0.41	-0.55	0.70
Function Parameters and Arguments	7.86	0.69	8.00	0.63	8.17	0.41	-0.31	0.28
Return Values	7.29	1.11	7.67	0.52	7.83	0.41	-0.55	0.70
PD - Declaration	7.43	2.57	8.50	0.84	8.67	0.82	-1.24	1.76
PD - Mathematical Operators	7.29	1.98	7.50	0.84	7.50	0.84	-0.21	1.14
Boolean Expressions	8.57	0.98	8.67	0.52	8.83	0.41	-0.26	0.57
Sequential Datatypes	8.00	1.41	8.67	0.52	9.00	0.00	-1.00	1.41
Variable Semantics	7.86	1.95	8.17	0.98	8.83	0.41	-0.98	1.54
Variable Assignment and Mutability	6.00	1.63	6.33	0.52	6.33	0.52	-0.33	1.12
Object-oriented Basics	4.14	2.67	5.25	1.26	5.25	1.89	-1.11	0.78
Error Interpreting	8.57	1.27	8.67	0.52	9.00	0.63	-0.43	0.64
External References	4.71	1.98	4.33	0.82	3.33	0.82	1.38	1.16
Debugger	4.57	2.37	4.50	1.64	4.20	0.84	0.37	1.53
AI Tools	2.71	2.63	3.00	2.10	2.00	0.71	0.71	1.92
External Data	7.43	1.81	8.00	0.63	8.17	1.17	-0.74	0.64
Conventions	6.29	2.50	6.17	0.75	6.17	0.98	0.12	1.51

Table 6: The mean and standard deviation of the importance rating by the experts from the first (n=7), second (n=6) and third (n=6) rating surveys. The difference is calculated by the first rating subtracted by the third rating. Primitive Datatypes is abbreviated as "PD" for space.

Difficulty Ratings	1st		2nd		3rd		Difference	
Concept Category	**Mean**	**SD**	**Mean**	**SD**	**Mean**	**SD**	**Mean**	**SD**
Control Flow Syntax	4.57	2.70	4.00	1.26	3.33	0.82	1.24	1.88
Sequential Execution	2.71	1.11	2.50	0.84	2.17	0.75	0.55	0.36
Conditional Execution	4.71	1.38	4.33	1.03	4.50	0.84	0.21	0.54
Looping	6.43	1.27	6.83	0.41	6.67	0.52	-0.24	0.76
Function Semantics	7.57	1.62	7.67	0.52	7.50	0.84	0.07	0.78
Function Parameters and Arguments	7.86	1.07	7.83	0.41	7.83	0.41	0.02	0.66
Return Values	6.43	1.27	6.83	0.75	7.00	0.63	-0.57	0.64
PD - Declaration	2.29	0.95	1.83	0.41	2.17	0.41	0.12	0.54
PD - Mathematical Operators	3.14	1.07	2.83	1.17	3.00	0.89	0.14	0.17
Boolean Expressions	4.43	1.40	4.67	0.82	5.00	0.63	-0.57	0.76
Sequential Datatypes	5.71	1.11	5.33	0.52	5.17	0.41	0.55	0.70
Variable Semantics	4.71	1.70	5.50	0.84	5.33	0.52	-0.62	1.19
Variable Assignment and Mutability	6.71	1.98	7.17	1.17	8.17	0.75	-1.45	1.22
Object-oriented Basics	6.43	3.46	7.67	1.21	7.80	1.30	-1.37	2.15
Error Interpreting	6.43	0.98	6.00	0.63	5.67	0.52	0.76	0.46
External References	4.86	0.38	5.00	0.00	5.00	0.00	-0.14	0.38
Debugger	6.14	1.35	5.50	0.55	5.60	0.55	0.54	0.80
AI Tools	2.71	1.80	3.17	1.17	3.40	0.55	-0.69	1.25
External Data	6.14	1.68	6.00	0.89	5.83	0.75	0.31	0.92
Conventions	3.57	1.40	3.50	0.84	3.33	0.82	0.24	0.58

Table 7: The mean and standard deviation of the difficulty rating by the experts from the first (n=7), second (n=6) and third (n=6) rating surveys. The difference is calculated by the first rating subtracted by the third rating. Primitive Datatypes is abbreviated as "PD" for space.

Ranking	**Concept**	**Aggregate**
1	Looping	25.8
2	Function Semantics	24.2
3	Function Parameters	22.9
4	Error Interpreting	21.4
5	Condtitional Execution	20.9
6	Return Values	20.7
7	Sequential Datatypes	20.5
8	Variable Semantics	20.4
9	Boolean Expressions	19.7
10	External Data	19.7
11	Variable Assignment	18.1
12	Control Flow Syntax	16.1
13	Object-oriented	14.7
14	Sequential Execution	14.5
15	Primitive Datatypes - Mathematical Operators	13.0
16	Primitive Datatypes - Declaration	12.8
17	Conventions	11.3
18	Debugger	9.9
19	External References	7.4
20	AI Tools	3.7

Table 8: Aggregate metric of importance and difficulty ratings from the final survey (n=6).

Appendix B

Exam Analysis

The variation between CS1 exams is illustrated in Table 9 by the mean and standard deviation of each concept category occurrence and use as main concept in final exams.

	Mean		SD	
Concept Category	**Occur**	**Main Concept**	**Occur**	**Main Concept**
Control Flow Syntax	57.5	3.4	28.0	6.4
Conditional Execution	50.3	30.0	14.0	10.4
Looping	61.8	21.5	5.9	11.7
Function Semantics	64.1	19.2	19.6	13.7
Function Parameters and Arguments	64.8	2.8	19.3	4.0
Return Values	61.3	0.7	22.5	1.6
Primitve Datatypes - Mathematical Operators	46.9	3.0	20.7	5.5
Boolean Expressions	48.8	2.1	16.8	3.4
Sequential Datatypes	69.6	33.5	8.6	10.6
Variable Assignment and Mutability	16.4	3.4	33.4	6.8
Object-oriented Basics	24.0	21.4	35.6	31.2
Error Interpreting	5.3	0.0	6.6	0.0
External References	11.8	0.1	19.0	0.2
Debugger	1.7	1.7	4.1	4.1
AI Tools	0.0	0.0	0.0	0.0
External Data	16.4	15.0	8.0	9.4
Conventions	10.5	6.2	11.8	8.0

Table 9: The mean and standard deviation of concept category occurrences and instances where the task was identified as the main concept being assessed.

Appendix C

Student Ratings

The concept categories and their descriptions detailed in Table 5 were rated by 55 CS1 and 88 CS2 students. Note that the concept category *Object-oriented Basics* was excluded from these surveys because it was not covered in their respective CS1 courses, thereby preventing these students from evaluating its difficulty and importance. Table 10 presents their mean ratings.

	Importance		**Difficulty**	
Concept Category	**CS1**	**CS2**	**CS1**	**CS2**
Control Flow Syntax†	8.7	9.02	4.74	4.67
Sequential Execution	8.6	8.49	4.06	4.78
Conditional Execution	8.66	8.77	5.38	4.00
Looping†	8.45	8.94	5.51	5.34
Function Semantics	7.98	8.43	4.75	5.72
Function Parameters and Arguments	8.13	8.04	4.42	4.92
Return Values	8.35	8.18	4.81	4.36
Primitive Datatypes - Declaration	7.83	8.04	3.48	3.18
Primitive Datatypes - Mathematical Operators	7.83	7.91	3.64	3.62
Boolean Expressions	8.22	8.22	3.92	3.78
Sequential Datatypes	7.73	7.74	5.22	5.37
Variable Semantics‡	7.91	7.75	3.84	4.21
Variable Assignment and Mutability‡	7.38	7.33	5.27	5.92
Error Interpreting	8.65	8.65	6.45	6.74
External References	7.37	7.55	5.02	5.08
Debugger	7.09	7.16	5.55	6.00
AI Tools	6.23	6.24	3.54	3.42
External Data†	6.61	6.03	6.5	6.50
Conventions‡	6.94	7.19	4.79	4.33

Table 10: The mean importance and difficulty ratings by the CS1 (n=55) and CS2 (n=88) students. "†" denotes significant difference of importance mean. "‡" denotes significant differences in difficulty mean.