A Proposal for a Programming Skills Framework Integrating Computational Thinking and Python Programming Concepts

Sébastien Hoarau, Christophe Declercq, Sophie Chane-Lune

A Proposal for a Programming Skills Framework Integrating Computational Thinking and Python Programming Concepts

Sébastien Hoarau
Christophe Declercq
Sophie Chane-Lune
sebastien.hoarau@univ-reunion.fr
christophe.declercq@univ-reunion.fr
sophie.chane-lune@univ-reunion.fr
Laboratoire d'Informatique et de Mathématiques
Université de La Réunion
Saint-Denis, France

Abstract

This paper presents an operational skills framework specifically designed for high school computer science teachers. This highly detailed framework enables the differentiation of students' abilities in two key areas: computational thinking skills and Python programming concepts. It is structured along two dimensions: first, six core skills — evaluation, modeling, anticipation, decomposition, generalization, and abstraction — and second, programming concepts related to data and control structures, as utilized in a programming language like Python. We describe the abilities at the intersection of each of these skills and provide for each ability a task model and a set of associated elementary tasks. We demonstrate various applications of this framework for teachers, including articulating learning objectives, selecting and categorizing tasks for students, and evaluating their skills. Finally, we discuss ongoing development efforts and the potential for international adaptation of the framework.

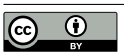
CCS Concepts

• **Social and professional topics** → **Computational thinking**; **K-12 education**; **CS1**; **Student assessment**.

Keywords

Computational thinking; Python; skills

ACM Reference Format:
Sébastien Hoarau, Christophe Declercq, and Sophie Chane-Lune. 2025. A Proposal for a Programming Skills Framework Integrating Computational Thinking and Python Programming Concepts. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2025), June 27-July 2, 2025, Nijmegen, Netherlands.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3724363.3729025



ITiCSE 2025, Nijmegen, Netherlands

ACM ISBN 979-8-4007-1567-9/2025/06
https://doi.org/10.1145/3724363.3729025

1 Introduction

The impact of learning programming on the development of problem solving skills has been extensively studied [11], though conclusive results remain limited. Since Jeannette Wing's seminal work [19], significant efforts have been made to define computational thinking (CT) precisely. CT is now widely recognised as a problem solving skillset akin to the cognitive processes required for computer programming. Although computational thinking is not synonymous with programming, programming can serve as a means to help students develop CT skills, which may then be applied across various scientific domains.

Our research, initiated in 2021, aims to support high school computer science teachers in implementing a skills-based approach to organize instruction and to assess student performance. The French computer science curriculum, established in 2019, incorporates this methodology and outlines specific skills in its preamble, such as modeling, algorithmic thinking, decomposition, abstraction, and generalization. However, the curriculum does not explicitly link these skills to the concepts and abilities described further in the document.

Recognizing that this curriculum was influenced by the evolving definition of computational thinking proposed by Selby and Woollard [17], we based our work on their framework, which includes the abilities to think abstractly, decompose problems, evaluate solutions, generalize ideas, and think algorithmically. While Selby and Woollard argue that "the definition may ensure that appropriate assessment tools can be developed which measure computational thinking skills", our initial challenge was to specify the programming context in which a given skill is evaluated.

For example, replacing a constant with a variable in a program does not necessarily indicate a student's mastering of generalization unless they also demonstrate the ability to add a parameter to a function. Similarly, a student cannot be said to grasp abstraction if they can encapsulate a process within a function but are unable to define a class. Furthermore, a student's modeling ability is inadequately assessed if they can only manipulate elementary-type variables but lack proficiency in utilizing data structures. These observations highlight the necessity of contextualizing the programming environment in which a skill is assessed, thus enabling the identification of different skill levels and providing a clearer pathway for evaluation.

We explore approaches that integrate computational thinking (CT) with computing concepts. Brennan and Resnick [6] identified seven core programming concepts—sequences, loops, parallelism, events, conditionals, operators, and data—that are widely applicable across Scratch projects and transferable to other programming contexts. They also outlined four computational practices: being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing. These practices closely align with the CT skills defined by Selby and Woollard. Given our focus on imperative, functional, and object-oriented programming in Python, we retain some of these concepts while replacing others to better suit our context.

Dagienė, Sentance, and Stupurienė [8] proposed a Two Dimensional Categorization System for Educational Tasks in Informatics. This framework specifies context through five domains: algorithms and programming, data structures and representations, computer processes and hardware, communication and networking, and interactions, systems, and society. Our work extends their approach by refining the programming and data structures contexts while focusing exclusively on these two areas.

Additional studies have examined the relationship between CT and Python programming. Bai, Wang, and Zhao [2] demonstrated that students taught using a problem-oriented learning model outperformed those taught using a lecture-and-practice model in a Python course on CT concepts. Similarly, Leticia Laura-Ochoa and Norka Bedregal-Alpaca [14] incorporated computational thinking practices to improve learning outcomes in an introductory Python programming course.

A literature review by Liu et al. [12] on computational thinking assessment identified three commonly used evaluation approaches: artifact analysis, CT assessment items, and interviews. They also proposed new methods involving visual attention and verbalization. Izu and Mirolo [9] investigated the benefits of code refactoring exercises. While their study does not explicitly situate itself within computational thinking, it demonstrates that students develop evaluation, generalization, and abstraction skills through this process. Additionally, Pasterk and Benke [15] compared CT with self-regulated learning, highlighting strong connections between these two reflective practices. Collectively, these studies provide valuable insights into the cognitive processes involved in programming.

Another motivation for our work was to better understand difficulties faced by novice programmers. Luxton-Reilly et al [13] have demonstrated how some complex assessments can be decomposed into atomic elements that can be assessed independently. They argue that this approach "may increase opportunities for novices to demonstrate what they can achieve, and may improve diagnosis of student difficulties". Our research aligns with this perspective, as we propose the use of both atomic and complex tasks to evaluate programming abilities, and to compare students' results across related assessments.

In this paper, we present our framework, focusing on the rationale behind our adaptation of established definitions of computational thinking. We also offer strategies to support computer science teachers in designing learning activities and conducting both formative and summative assessments of their students' skills.

2 Proposed framework

Our approach integrates programming-related concepts with computational thinking (CT) skills. Drawing from various definitions of skills, abilities, and tasks [16], we define a skill as a collection of abilities that an individual can leverage to successfully perform a complex task.

2.1 Computational thinking skills

We propose the following verbs to represent computational thinking (CT) skills:

Evaluate : The skill to mentally assign a value (e.g., result, type) to a given program. It's the most elementary skill as it only involves ability to read a given program in order to evaluate its result or type.

Model : The skill to represent available information and the information to be computed using appropriate data structures. It's related to abstraction as programmers have to abstract real word. Modeling is the first step towards programming and may lead to a concrete program without abstract data structures or abstract control structures.

Anticipate : The skill to adopt the perspective of a programmer tasked with describing the sequential, repetitive, or conditional composition of instructions in an algorithm, before its execution. It's the most difficult task for a novice programmer : "programming a solution to a given problem is not solving it, it's writing how a computer could solve it". Anticipating is the main challenge for beginners, evolving from direct manipulation [10] to programming.

Decompose : The skill to break down a complex problem into a set of simpler sub-problems equivalent to the initial problem. In both case decomposition and sequential decomposition, this process handles problem descriptions and may be expressed in natural language. The decomposition process does not necessarely generate specific language constructions and may be visible only with comments in the final program.

Generalize : The skill to infer a general problem from a specific instance and to identify recurring patterns in processing or data within a particular problem. As already mentioned, in the context of programming, generalization consists in replacing constants by variables or parameters and sequences by loops. It's also generalizing a binary operator as an operator over lists.

Abstract : The skill to disregard non-essential details and create solutions where the implementation details are obscured by using an appropriate interface. We use abstraction as [1] who described, more than thirty years ago, ways to abstract with procedures or data.

We chose to include "model" as a distinct computational thinking (CT) skill because we observed that beginner students often grasp modeling relatively quickly but struggle significantly with abstraction. This term was already present in Wing's original definition: "modeling the relevant aspects of a problem to make it tractable". While Selby and Woollard acknowledged the prevalence of this term in their literature review, they ultimately subsumed it under the broader skill of "abstraction". However, we found it necessary

Skills Select All	(E) Evaluate Select All	(M) Model Select All	(A) Anticipate Select All	(D) Decompose Select All	(G) Generalize Select All	(X) Abstract Select All
(01) Programming with Expressions Select All	(E01) Evaluate the value and type (integer, string or Boolean) of an expression with variables, constants and operators	(M01) Model the information available/to be calculated by elementary variables (integer, string and Boolean)	(A01) Anticipate writing of an expression with variables, constants and operators	(D01) Decompose a complex expression by identifying one or more intermediate variables	(G01) Generalize an expression by replacing a constant with a variable	
(02) Programming with assignments and sequences Select All	(E02) Evaluate the effects of running a program with a sequence of assignments	(M02) Model the information to be calculated by one or more intermediate variables	(A02) Anticipate writing of a sequential treatment to obtain a specified result, the method being given	(D02) Decompose complex processing by identifying one or more intermediate variables		
(03) Programming with bounded loops Select All	(E03) Evaluate the effects of running a program with a bounded loop	(M03) Model with one or more variables the information to be changed at each loop	(A03) Anticipate writing of a bounded loop to obtain a specified result, the method being given	(D03) Decompose complex processing by identifying a bounded loop	(G03) Generalize sequential treatment by replacing it with repetitive treatment	
(04) Programming with unbounded loops Select All	(E04) Evaluate the effects of running a program with an unbounded loop	(M04) Model with one or more variables the information to be changed at each repetition	(A04) Anticipate writing of a repetitive treatment to obtain a specified result, the method being given	(D04) Decompose complex processing by identifying an unbounded loop		
(05) Programming with alternative or conditional constructions Select All	(E05) Evaluate the effects of running a program with alternative or conditional construction(s)	(M05) Model the condition of an alternative or conditional by a boolean variable	(A05) Anticipate writing of an alternative or conditional treatment to obtain a specified result, the method being given	(D05) Decompose complex treatment in several cases using alternative or conditional constructions		
(06) Programming with functions Select All	(E06) Evaluate the result of a function call with given parameters	(M06) Model a function processing by specifying its parameters	(A06) Anticipate writing of a function, the parameters being specified	(D06) Decompose complex processing using function composition	(G06) Generalize a treatment, by defining a function with parameters or by adding a parameter to a function	(X06) Abstract a treatment by a function definition with parameters and comments
(07) Programming with recursive functions Select All	(E07) Evaluate the result of a recursive function call	(M07) Model treatment with recursive function by specifying its parameters	(A07) Anticipate writing of recursive function, parameters being specified		(G07) Generalize recursive function with function as parameter	
(08) Programming with tuples Select All	(E08) Evaluate an expression to access tuple information	(M08) Model complex information with a tuple of basic information	(A08) Anticipate access to basic tuple information			(X08) Abstract complex data using tuples
(09) Programming with records Select All	(E09) Evaluate an expression to access information from a record	(M09) Model complex information with a record	(A09) Anticipate access to basic information from a record			(X09) Abstract complex data using records
(10) Programming with arrays - construction and access Select All	(E10) Evaluate an expression or effects of running a program using an array	(M10) Model a series of information by an array	(A10) Anticipate the treatments to program access to elements of an array by their indices and modify them		(G10) Generalize writing an array using comprehension	(X10) Abstract complex data using arrays
(11) Programming withs arrays - iterate Select All	(E11) Evaluate the effects of running a program iterating on an array	(M11) Model an array processing by choosing to iterate on indices or elements	(A11) Anticipate a linear array iteration		(G11) Generalize a treatment by applying it to all elements of an array	
(12) Programming with dictionaries - construction and access Select All	(E12) Evaluate an expression or effects of running a program using a dictionary	(M12) Model information association by dictionary	(A12) Anticipate treatment to add or modify a combination to a dictionary			(X12) Abstract complex data using dictionaries
(13) Programming with dictionaries - iterate Select All	(E13) Evaluate the effects of running a program iterating over a dictionary	(M13) Model a dictionary processing by choosing to iterate on keys, values or pairs	(A13) Anticipate a treatment that requires iteration on a dictionary			
(14) Programming with interfaces and implementations Select All		(M14) Model data and processing by specifying a data structure through its interface	(A14) Anticipate implementation of a data structure			(X14) Abstract a processing using a data structure to be specified.
(15) Programming with objects and classes Select All	(E15) Evaluate the creation of an instance, the invocation of methods and access to attributes of an object	(M15) Model data and processing by specifying a class, attributes and methods	(A15) Anticipate treatments by implementing class methods	(D15) Decompose data and treatments into classes	(G15) Generalize a class by adding an attribute or method	(X15) Abstract a set of data and treatments by encapsulating them in a class as attributes and methods

Figure 1: Programming skills framework

to treat modeling as a separate skill, as it enables us to differentiate between beginners' ability to model and their ability to abstract.

Abstraction and generalization are often confused because both can be implemented by functions. However, they are two distinct processes, with abstraction primarily hiding details, while generalization increases the scope of the proposed solution.

These cognitive skills are required at different stages of a programming task. For instance, when reading a given program, students primarily engage in the "evaluate" skill. In contrast, when starting with a blank page, the first step is to "model". The "anticipate" skill is central to the act of programming itself, as it involves algorithmic thinking. If the problem is too complex, students must "decompose" it into more manageable parts before attempting a solution. "Generalize" and "abstract" are higher-order skills that typically challenge beginners; tasks requiring these skills often involve rewriting programs, to transition from a concrete or specific implementation to a more abstract or generalized one.

Referring to Bloom's taxonomy [4], the "evaluate" skill corresponds to the comprehension level, the "model", "anticipate" and "decompose" skills align with the application level, and the "abstract" and "generalize" skills belong to the analysis and synthesis levels.

In our 2D framework, each column represents a computational thinking skill. For example, the "model" skill encompasses a range of abilities, such as modeling with variables, tuples, records, arrays, or objects. A student demonstrates mastery of this skill when they can successfully model a problem by selecting an appropriate data structure.

2.2 Programming concepts

The concepts addressed in this framework are derived from the programming and data structures outlined in the French high school curriculum. We deliberately limit our scope to these concepts to better understand the challenges faced by novice programmers. The framework is instantiated using Python language constructs, although these constructs and data structures are common to most general-purpose imperative, functional, and object-oriented programming languages. The programming concepts covered in this framework include expressions, assignments, sequences, bounded loops, unbounded loops, conditional or alternative structures, functions, recursive functions, tuples, records, arrays, dictionaries, interfaces and implementations, objects, and classes.

These concepts are equally relevant for an introductory university level computer science course (CS1).

Compared to those concepts identified by Brennan and Resnick [6], our framework provides a more detailed classification of sequences, loops, conditionals, operators, and data. However, we deliberately exclude concepts such as parallelism and events, as they are not covered in the French high school curriculum. Compared to concepts in CS1 as identified from literature and learning

outcomes [13], our framework includes most of fondamental topics, with the exception of pointers, memory management and abstract programming thinking. In the French high school curriculum, this last concept is classified as an algorithmic concept rather than a programming one.

In our 2D framework, each row represents a programming skill. For instance, the skill "Programming with functions" encompasses six abilities: evaluating, modeling, anticipating, decomposing, generalizing, and abstracting using functions.

2.3 Skills, abilities and elementary tasks

We have defined a "task model" for each ability. A task model is associated with at least three tasks that serve as specific instances of the model. Figure 2 illustrates this approach for the ability "Generalize sequential treatment by replacing it with repetitive treatment", showing the associated task model (in gray) and three task examples. The complete framework encompasses 65 abilities and over 200 proposed tasks.

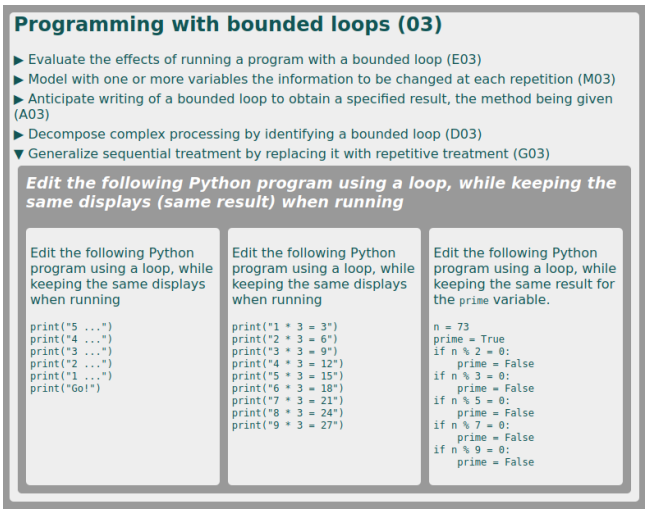


Figure 2: Abilities, task models and tasks

As we will discuss later, these tasks serve both as training and diagnostic assessment for students.

3 A framework for high school computer science teachers

The primary goal of this skills framework is to assist teachers in aligning educational objectives, learning tasks, and student assessments. Educational objectives are defined in curricula.

In the French high school computer science curriculum (referred to as "Numérique et Sciences Informatiques"), the objectives center on developing computational thinking skills. However, teachers have some flexibility in designing a pedagogical progression that ensures these objectives are achieved within the annual program.

In the first year (age 15), the curriculum focuses on six foundational programming skills: programming with expressions, assignments and sequences, bounded and unbounded loops, conditional or alternative structures, and functions. The second-year curriculum introduces all remaining concepts, except recursive functions and object-oriented programming, which are covered in the final year.

3.1 Selecting elementary tasks

For introductory learning, teachers can assign elementary tasks aligned with each ability defined in the framework. These tasks can be introduced at the beginning of a course. Teachers may also use the framework to select tasks either from a single row (corresponding to a programming skill) or a single column (corresponding to a computational thinking skill). This approach allows for targeted revisions, whether focusing on a specific programming concept or on a computational thinking practice. Through the online tool provided alongside the skills framework (see Figure 1), teachers can select the desired skills or abilities to generate customized task lists for students.

This approach also promotes diversity in task design. We observed that computer science textbooks for Python programming almost exclusively include tasks structured as: "Write a Python function that..."

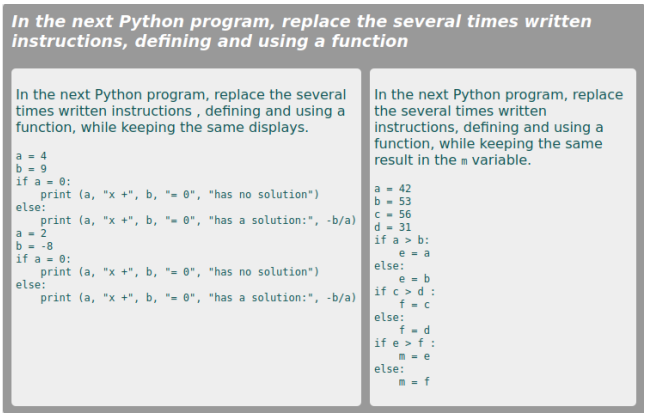


Figure 3: Tasks for G06 (Generalize with a function)

In contrast, our framework introduces a broader variety of tasks (see Figure 3), which address common challenges faced by beginners when learning to use functions. Tasks such as generalizing or refactoring code for instance, require significant practice. Students must be systematically trained to master these types of tasks.

3.2 Categorizing complex tasks

Problem-oriented learning model has proven to be an effective approach for teaching computational thinking through programming. To enable teachers in implementing such a model, it is essential to provide a method for categorizing complex tasks encountered during problem-solving in alignment with our framework. Categorizing a complex task involves listing all the abilities required for its completion. Given that a complex task often has multiple valid solutions, this list of abilities is not unique. This approach helps teachers define the necessary prerequisites before engaging students in problem-solving activities. This process is also crucial

for providing remediation to students who have not yet mastered the foundational skills.

Consider the following problem: Calculate the number of balls in a tetrahedron of height 22, knowing that the tetrahedron shown in Figure 4 has a height of 4.

Figure 4: Calculating the number of balls in a tetrahedron

If students lack prior knowledge of mathematical series, they must first model the available information and the information to be calculated. This involves using the ability to model with elementary variables (M01). Suppose one student names `height` as the height of the tetrahedron and `total` as the total number of balls. To solve the problem, the student must decompose the task by identifying a bounded loop (D03) and model the information that will change in each iteration (M03), then implement the loop (A03). While attempting to anticipate writing a sequential treatment (A02) to compute the next value of `total`, the student may encounter a difficulty : the number of balls in a tetrahedron of height 4 cannot be directly derived from the number of balls in a tetrahedron of height 3.

The student must then consider modeling an intermediate calculation using an intermediate variable (M02), which represents the number of balls per floor. Suppose the student defines `floor` as the number of balls on a single level. With these abilities (M01, M02, M03, A02, A03, D03), a solution can be constructed :

```
floor = 0
total = 0
for height in range (1, 23):
    floor = floor + height
    total = total + floor
print(total)
```

Some students may model the problem using arrays (M10) to compute the total number of balls (A11) before summing them. Others may generalize the process by creating a function (G06) that takes `height` as a parameter. It is crucial for the teacher to conduct this analysis beforehand to identify the necessary prerequisites and the abilities that may be assessed based on student performance.

3.3 Assessing students abilities and skills

Our framework distiguishes between abilities - which can be evaluated through elementary tasks - and skills, which can only be assessed in complex tasks contexts. Within a learning management system such as Moodle, abilities can be assessed using achievement conditions. When a student successfully completes a task, the corresponding ability is recorded. Assessing abilities is primarily used as a diagnostic tool, helping teachers identify challenges faced by students and consider appropriate remediation strategies.

Since a skill is defined as a "set of abilities that a subject may mobilize to perform a complex task", it cannot be assessed simply by a student's success in completing all tasks associated with each ability. A skill is more than just the sum of its abilities. Mastering all abilities is a necessary, but not sufficient, condition for skill acquisition.

Assessing skills requires complex tasks specifically designed to test each ability in a realistic context. For instance, to assess the "model" skill, a teacher might propose a complex situation to model (e.g., a game board) and evaluate whether each student provides a coherent model using appropriate data structures.

Different skill levels have been proposed: beginner, intermediate, advanced, and expert. Within our framework, skill levels can be distinguished using the 2D structure of the framework. Programming skills can be used to define levels for computational thinking skills, and computational thinking skills can define levels for programming skills.

For example, a beginner might be able to model using only elementary variables, while an intermediate programmer would use data structures such as tuples, records, and arrays. An advanced programmer would model using objects, while an expert programmer would use the best data structure to optimize a specific criterion. This creates a descriptive scale for assessing computational thinking skills.

Similarly, different levels of programming skills (e.g., programming with functions), can be defined, reflecting the increasing complexity of computational thinking skills: evaluating a given program is the most basic ability and may be assessed at the beginner level. Modeling, anticipating, and decomposing are core programming abilities assessed at the intermediate level. Generalizing and abstracting are higher-level abilities, assessed at the advanced or expert level. This provides a descriptive scale for assessing programming skills.

4 Experiments and future works

Initial experiments were conducted in France, as our framework was developped based on the French curricula. We now aim to collaborate with teachers from diverse educational systems to test the framework's applicability in different countries where computer science is taught both in high school and university.

4.1 In France

Experiments began in September 2023 in a high school classroom (age 16) and in a CS1 course at the university (age 18). A positioning test, based on elementary tasks from the first six lines of our framework, was administered to students. Results show that the best outcomes were obtained for the "evaluate" skill. Lower scores were observed for "model", "anticipate", and "decompose". The "generalize" and "abstract" skills yielded near-zero scores. The findings were presented at a French-speaking conference and confirmed our initial hypothesis regarding the scaling of complexity in computational thinking skills [7]. Additionally, the analysis of CS1 students' results revealed a significant gap between students who had studied computer science in high school and those who had not.

Since December 2024, the stabilized framework has been incorporated into the continuing education of computer science teachers. While data analysis is ongoing, preliminary results indicate that teachers are able to adopt the framework to categorize complex tasks. Further research is needed to analyze changes in teaching practices related to skill-based learning and assessment.

4.2 Internationalization

Since its initial version, our framework has been made available in French on a GitLab server, where all tasks, task models, and skills are stored in a structured file system using markdown files. This organization was designed to provide various views of the repository from a single source, allowing for both synthetic and detailed views as well as task selection tools. Naturally, this work has been developed collaboratively, with over one hundred commits recorded in the repository.

The framework has been internationalized, particularly for presentation at an international conference and then allowing collaborations and comparisons within different educational systems. Internationalization involved extracting from program files all language specific strings and translating all markdown files into English. With these technical preparations complete, we are now ready to propose this work to teachers in English-speaking schools, particularly those involved in Computing at School in the UK or K-12 programming courses in the US. Additionally, we may translate the framework into other European languages with support from native-speaking teachers.

4.3 Future works

We have collected learning data from a high school classroom in collaboration with a teacher who is part of our research team. Although the initial results can be considered encouraging, it is essential to expand the experiment on a larger scale by involving a team of trained teachers who will implement the skill-based approach in their classes. This will allow us to test our research hypothesis regarding the operationality of the framework. An experimental protocol is currently being developed to measure teachers' adoption of the framework and the impact on their classroom practices, particularly in terms of alignment between learning objectives, assigned tasks and student assessments.

Our second research hypothesis, which is also being tested, is the framework's consistency. We define consistency by its ability to produce coherent results across different assessments of the same student's work. Specifically, we consider results to be inconsistent if a student demonstrates, within a complex task, abilities deemed essential during the categorization of this task but fails to exhibit these same abilities during an assessment based on elementary tasks. During a previous experiment with an earlier version of the framework, bounded and unbounded loops were treated as a single concept. While students had the necessary abilities to solve complex tasks involving bounded loops, they struggled with elementary tasks focused solely on unbounded loops. This discrepancy led us to separate the two concepts to eliminate inconsistency.

These two areas of research require the collection of teaching data with minimal additional workload for teachers. To achieve this, we are currently designing a plugin for the Moodle Learning Management System (LMS). This dedicated plugin will allow teachers to assign tasks to students, identify the relevant abilities involved in these tasks, and evaluate student submissions based on these abilities. The plugin will also distinguish between abilities addressed in elementary tasks and those mobilized by students in the context of complex tasks. By analyzing learning traces collected through this system, we aim to refine the skill validation rules, ensuring they accurately reflect a student's mastery of a given skill at a specified level.

5 Conclusion

We have proposed an innovative skills framework that integrates computational thinking skills with Python programming concepts. The primary goal is to provide educators with a set of tools that focuses on programming concepts outlined in the official French high school curriculum. However, drawing upon work on computational thinking and our initial presentations within the French-speaking academic community, we believe this framework is not inherently specific to the French education system. Therefore, we propose its application in English-speaking contexts and the internationalization of the associated software environment. Partial adoption of the framework is also feasible, especially when the programming curriculum covers only a subset of the initial components outlined in the framework.

Since the programming concepts used in our framework are not exclusive to Python language, we could study extension to other imperative language (Java, C++). It is to allow this generalization that we have, for example, chosen to separate the concepts of records and dictionaries, concepts which are confused in Python. Additionally we plan to investigate how the framework can support mastery learning [3] especially within "parallel locked" and "spiral" models [18].

Our current work is organized around two complementary directions. The first, more theoretical, seeks to establish the operationality and consistency of the framework by analyzing data collected from teachers who have implemented it. The second, more practical, focuses on developing a software environment that supports teachers in categorizing tasks and evaluating student performance. These two dimensions are closely connected, as the second facilitates the collection of data for the first.

We also still have to compare the use of our framework for assessment with known works on the assessment of programming skills at the end of high school or at the beginning of CS1 courses. For example, studying "Nordic Prior Knowledge Test in Programming" [5] could help us to understand which abilities are really assessed at the end ok K12 programming instruction. Such a test would also be very useful in France, to understand the impact of CT introduction in high school, on students' skills when entering in university.

For this research to progress effectively, we are open to collaborating internationally with educators and researchers interested in experimenting with the framework and its associated tools. All our resources are published under open licenses and are publicly available online :
`https://iremi974.gitlab.io/rcp/index.html?lang=en`

References

[1] Harold Abelson and Gerald Jay Sussman. 1996. *Structure and interpretation of computer programs.* The MIT Press, Cambridge.

[2] Hongquan Bai, Xin Wang, and Li Zhao. 2021. Effects of the Problem-Oriented Learning Model on Middle School Students' Computational Thinking Skills in a Python Course. *Frontiers in Psychology* 12 (Dec. 2021), 1–14. doi:10.3389/fpsyg.2021.771221 Publisher: Frontiers.

[3] Benjamin Samuel Bloom. 1968. Learning for Mastery. *Evaluation Comment* 1, 2 (1968), 12 pages. https://api.semanticscholar.org/CorpusID:140387417

[4] B. S. Bloom, M. B. Engelhart, E. J. Furst, W. H. Hill, and D. R. Krathwohl. 1956. *Taxonomy of educational objectives. The classification of educational goals. Handbook 1: Cognitive domain.* Longmans Green, New York.

[5] Sondre Bolland, Andreas Haraldsrud, Siri Jensen, Filip Strömbäck, Arne Styve, Erlend Tøssebro, Eirik Valseth, and Torstein Strømme. 2024. The Nordic Prior Knowledge Test in Programming: Motivation, Development and Preliminary Results. *Norsk IKT-konferanse for forskning og utdanning* 4 (11 2024), 14 pages. doi:10.5324/nikt.6216

[6] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In *annual American Educational Research Association meeting* (Vancouver, Canada). AERA, Cambridge, Massachusetts, 25 pages.

[7] Sophie Chane-Lune, Christophe Declercq, and Sébastien Hoarau. 2024. Un référentiel de compétences en programmation pour identifier les difficultés des débutants et différencier les activités. In *Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC. Volume 1*, Kim Mens and Olivier Goletti (Eds.). HAL, Louvain-La-Neuve, Belgium, 53–63. https://hal.science/hal-04482126

[8] Valentina Dagienė, Sue Sentance, and Gabrielė Stupurienė. 2017. Developing a Two-Dimensional Categorization System for Educational Tasks in Informatics. *Informatica* 28, 1 (2017), 23–44. doi:10.15388/Informatica.2017.119

[9] Cruz Izu and Claudio Mirolo. 2024. Asking Students to Refactor their Code: A Simple and Valuable Exercise. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 73–79. doi:10.1145/3649217.3653546

[10] Ivan Kalaš and Klara Horvathova. 2021. Programming Concepts in Lower Primary Years and Their Cognitive Demands. In *IFIP Advances in Information and Communication Technology (Digital Transformation of Education and Learning - Past, Present and Future, Vol. AICT-642)*, Don Passey, Denise Leahy, Lawrence Williams, Jaana Holvikivi, and Mikko Ruohonen (Eds.). Springer International Publishing, Tampere, Finland, 28–40. doi:10.1007/978-3-030-97986-7_3 Part 1: Digital Education Across Educational Institutions.

[11] D. Midian Kurland, Roy D. Pea, Catherine Clement, and Ronald Mawby. 1986. A study of the development of programming ability and thinking skills in high school students. *Journal educational computing research* 2(4) (1986), 429–458. https://telearn.hal.science/hal-00190539

[12] Ruohan Liu, Feiya Luo, and Maya Israel. 2021. What Do We Know about Assessing Computational Thinking? A New Methodological Perspective from the Literature. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (Virtual Event, Germany) *(ITiCSE '21)*. Association for Computing Machinery, New York, NY, USA, 269–275. doi:10.1145/3430665.3456380

[13] Andrew Luxton-Reilly, Brett A. Becker, Yingjun Cao, Roger McDermott, Claudio Mirolo, Andreas Mühling, Andrew Petersen, Kate Sanders, Simon, and Jacqueline Whalley. 2018. Developing Assessments to Determine Mastery of Programming Fundamentals. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports* (Bologna, Italy) *(ITiCSE-WGR '17)*. Association for Computing Machinery, New York, NY, USA, 47–69. doi:10.1145/3174781.3174784

[14] Leticia Ochoa and Norka Bedregal-Alpaca. 2022. Incorporation of Computational Thinking Practices to Enhance Learning in a Programming Course. *International Journal of Advanced Computer Science and Applications* 13 (01 2022). doi:10.14569/IJACSA.2022.0130224

[15] Stefan Pasterk and Gertraud Benke. 2024. Computational Thinking for Self-Regulated Learning. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 640–645. doi:10.1145/3649217.3653565

[16] Margarida Rodrigues, Enrique Fernández-Macías, and Matteo Sostero. 2021. *A unified conceptual framework of tasks, skills and competences.* JRC Working Papers Series on Labour, Education and Technology 2021/02. European Commission, Seville. https://hdl.handle.net/10419/231348

[17] Cynthia Selby and John Woollard. 2013. Computational thinking: the developing definition. https://eprints.soton.ac.uk/356481/ Num Pages: 6 Publisher: University of Southampton.

[18] Claudia Szabo, Miranda C. Parker, Michelle Friend, Johan Jeuring, Tobias Kohn, Lauri Malmi, and Judithe Sheard. 2025. Models of Mastery Learning for Computing Education. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 1092–1098. doi:10.1145/3641554.3701868

[19] Jeannette M. Wing. 2006. Computational Thinking. *Commun. ACM* 49, 3 (2006), 33–35.