ISSN: 2473-4901
v12 n6818

Using Artificial Intelligence in Teaching and Learning: A Case Study of an Undergraduate Computer Programming Course

Peter Y. Wu
wu@rmu.edu
Computer & Information Systems Department
Robert Morris University
6001 University Blvd, Moon, PA 15108, USA

Abstract

The rapid emergence of artificial intelligence (AI) challenges higher education to rethink traditional teaching and learning strategies. We did an exploratory case study of an undergraduate computer programming course in which we integrated AI by encouraging students to utilize it for programming assignments and leveraging it ourselves to generate testing materials. The students were guided to seek AI assistance not only for explaining specific programming constructs but also for algorithm design and code generation. Our pedagogical goal was to foster active engagement by having students read, edit, and understand AI-generated code, shifting the focus toward software design and program integration. To ensure accountability, programming assignments required comprehensive pseudo-code descriptions of the algorithms and explanations of intricate coding strategies within the comments. Besides programming assignments, a substantial portion of the course grade relied on in-class quizzes. AI was utilized to help generate quiz questions that required reading and analyzing short program segments. These questions required meticulous manual review and modification to ensure appropriateness for use. The quizzes were open-book and open-notes but the students were prohibited from using AI. The pedagogical intent is to balance AI assistance in the assignments to learn program design and software integration while testing the students' understanding of programming features strictly without AI. Additionally, a survey was administered to gauge student perceptions of AI utility. As an exploratory study, the results are hardly conclusive, but it suggests a practicable approach to incorporate AI into teaching computer programming.

Keywords: programming education, artificial intelligence, AI, pedagogy, reading and writing code.

Using Artificial Intelligence in Teaching and Learning: A Case Study of an Undergraduate Computer Programming Course

Peter Y. Wu

1. INTRODUCTION

The rapid emergence of artificial intelligence (AI) has transformed teaching and learning practices across higher education (Webb et al., 2021; Ng et al., 2025; Raihan et al., 2025). Many educators express concern that students may commit academic dishonesty by submitting AI-generated work without genuinely comprehending the material (MacNeil et al., 2023). However, because AI tools have become readily accessible, they offer valuable on-demand assistance to both students and instructors. This technological evolution presents both unique instructional opportunities and novel challenges for educators (Prather et al., 2024; Azoulay et al, 2025). Furthermore, as industries increasingly seek to hire graduates proficient in AI collaboration, academia must proactively embrace these tools. While some in the information technology sector believe AI may eventually replace entry-level programmers by generating code directly from technical specifications, we argue that information systems (IS) education must adapt rather than avoid confrontation (Frydenberg, 2026). Instructors need to rethink instructional strategies for computer programming (Zviel-Girshin, 2024; Philbin, 2025).

Our case study focuses on an introductory Java programming (Gosling et al., 2025) course. We share our experiences integrating AI into the curriculum, where we guided students to use AI for syntax queries, algorithm design, code generation and debugging. By encouraging students to critically read and edit AI-generated code, we shifted the paradigm from rote coding to active code analysis. To enforce academic integrity and deeper understanding, we implemented strict submission requirements, such as embedding pseudo-code description into comments and many detailed requirements as well. Concurrently, AI was used to assist in generating short, code-reading assessment questions for in-class quizzes, which students must complete without AI assistance. Combined, programming assignments and in-class quizzes constituted the grading components of the course. Finally, an end-of-semester survey was conducted to investigate student usage patterns and attitudes toward AI. This paper shares the experience of our exploratory study.

2. LITERATURE REVIEW

Learning to program requires the students to exercise their mental model of computation for problem solving, but also the need to deal with the additional conditions imposed by the programming language and the tools in the development environment. The richness of necessary concepts makes the learning process prone to cognitive overload (Mayer 1981).

Generative AI has become a wonderful extension to internet search tools for students of computer programming. The students doing their programming assignments often need to look up details of specific programming language features they plan to use. They may now have easy all-time access to the explanation of these features with generative AI assistance (Hadi et al., 2023; Imran & Almusharraf, 2024). These tools combined with the modern text editors and integrated development environments (IDEs) have offered great help to lighten up the cognitive load in learning how to program (Marron, 2024). On critical thinking for problem solving, however, generative AI is a double-edged sword. AI tools can offer good guidance in program design and can provide complete solutions for most elementary programming problems. As educators, we are reasonably concerned about students using AI to complete their programming assignments without tackling the necessary critical thinking role. Detection strategy of AI generated code does not work well since there are often variations in AI generated solutions (Finnie-Ansley et al., 2022). Further, solutions generated by AI tools may also apply sub-optimal solutions and practice a poor style (Chen et al., 2022). But if the students use the AI tool for guidance in program design, and are encouraged to read and analyze the generated code segments, it may serve as a good, knowledgeable partner in pair programming practice (Sentance, 2019; Philbin, 2025).

Apart from guiding the students to use AI in learning computer programming, there is also much work in applying AI to reorganize teaching (Avouris et al., 2025). While many use AI to design and generate teaching materials, many also apply AI in the assessment of student performance (Garzon 2025).

3. SYLLABUS AND TOPICS

As an introductory undergraduate course, the curriculum begins with the fundamental procedural programming concepts: primitive data types, values and variables, literals, assignment statements, conditional logic (if-else, switch) and loop constructs and arrays. To cultivate objects early in system thinking, we also introduce the Java run-time storage environment, teaching the students to differentiate stack space, heap space, and static space, high lighting that objects of abstract data types are kept in the heap space. More advanced topics include sorting and searching, text file processing, objects and classes, and exception handling. Table 1 below outlines the topics in the course syllabus.

Topic 1	**Elementary concepts** Data types, literals, values and variables Assignment statements, expressions, operators and operands Console I/O
Topic 2	**Control Structures** Selection structures: switch, if-else Loop constructs: while, do-while, for The stack space and scope of variables
Topic 3	**String Processing** The heap space for objects String structure and immutability, string comparison Text manipulation
Topic 4	**Array Processing** Array object structure: declaration and initialization, indexing Array attributes and methods Fundamentals of sorting and searching
Topic 5	**Methods** Parameters and arguments, call-by-value semantics Use of methods, overloading and resolution Recursion and the call stack
Topic 6	**The Programming Process** Top-down design and bottom-up implementation Drivers and stubs for software integration Compile-time and run-time errors
Topic 7	**Objects and Classes** Class - an Abstract data type, constructors and methods Public/private and static/instance members The static space and the Java run-time environment
Topic 8	**Exception Handling** Concepts of exception handling try-catch-finally, throws and throw Checked and unchecked exceptions, the Exception hierarchy

Table 1. Topics in the Course Syllabus

The lectures and reading assignments bring the students into the topics of the curriculum. Student performance will be graded by nine programming assignments and four in-class quizzes. The students are guided to engage with AI tools to work on the programming assignments, integrating code segments generated by AI to complete the programming assignments. The in-class quizzes are open book and open notes, but the students are not allowed AI assistance. The limited time of the quizzes makes AI assistance not very practicable, but the invigilator at the quiz prohibits it, too. The next two sections will discuss further details of our approaches in the programming assignments and the in-class quizzes.

4. PROGRAMMING ASSIGNMENTS

We have 10 programming assignments to progressively reinforce the theories and concepts covered in class. Table 2 below lists the 10 programming assignments (A0 to A9; A0 is not counted for credit.), along with the topic of exercise and a general description of the assignment.

A0	Hello World!	An exercise to create a program: compilation and execution. Assignment not counted for credit.
A1	Console I/O: buying pens	Dialog sequence using console input and output, for the purchase of pens, paying taxes and calculation of total. Option to use JOptionPane methods instead of console i/O.
A2	Selection: date to day of week	To compute day of week from a date given by year, month and date. Option to use the switch or if-else selection control structures.
A3	Loop Control: read and calculation from a stream	To read from a stream of numbers and instantly calculate statistics on the fly including number of numbers, sum, average, and standard deviation.
A4	More Loop Control: guess a random number	For the user to guess a randomly generated integer from 0 to 100, reporting whether the number is larger or smaller every time if not guess right, allowing up to 7 guesses.
A5	Array and Loops: find the primes	To find all the prime numbers up to a given limit: must use an array to implement the Siever of Eratosthenes to filter out the multiples less than or equal to the limit.
A6	Simple Sorting: insertion and bubble	Using methods to implement two simple sorting algorithms: insertion sort and bubble sort, including measurement of CPU time used in nano-seconds.
A7	Advanced Sorting: merge	Implement also merge sort and text file input to compare performance of sorting tens to hundreds of thousands of numbers, with an introduction to time complexity analysis.
A8	Text Processing	Text file input/output: to process a text file to remove punctuation marks, to prepare for extraction of tokens.
A9	Symbol Table of Objects	Text file input/output: to process a text file to extract tokens for the construction of a symbol table, using objects in a data structure.

Table 2. The 10 Programming Assignments

We explicitly encouraged the students to utilize generative AI tools – primarily Gemini 1.5 Pro (Imran & Almusharraf, 2024) and ChatGPT (Hadi et al., 2023) which are freely available on the web – as learning aids. In class, we demonstrated top-down approach to algorithm design. For bottom-up approach to implementation, when it was not reasonably clear how to implement a segment, we guided the students to engineer the prompts to AI to generate the Java code segments. The students' primary task then shifted to software integration: assembling disparate code segments into a cohesive, bug-free program. The students learned to write code by reading and refactoring existing code. Our idea is for the students to first learn the concepts from lectures and reading assignments, and to follow up with practicing to write code from reading and editing code (Kernighan & Ritchie, 1978).

We were concerned that some of the students might turn the entire programming assignment to AI to get the completed solution for submission without understanding the concepts or learning the coding details. To mitigate the risk there, we implemented the following grading criteria.

- **Mandatory Pseudo-code:** The submitted program must include a step-by-step pseudo-code description of the algorithm in the comments. Failure to provide it could result in an in-person code defense interview with the grader. This requirement started with A2.
- **Detailed Documentation:** Major logic blocks and intriguing code components require contextual explanation in the comments.
- **Best Practices:** Variable names need to be appropriate in the context of the assignment, and adherence to minimizing variable scope (declared within the closest usage block).

There might also be other more subtle requirements for each programming assignment, and the students would have to work on the program code to meet. Such an example might be in A2, use of arrays was not allowed in the program implementation.

Assignment Example: The Random Number Guessing Game
To illustrate our approach, consider an assignment (A5) after learning different types of loop control structures. The students were tasked to develop a program which generates a random integer from 1 to 100 inclusive, allowing the user up to 7 attempts to guess it. The program should state whether the target number is larger or smaller after each incorrect guess.

In top-down design, the students would break down the problem into smaller modules. The first module would generate a random integer number between 1 and 100. The students could easily have AI generate the code segment for that. However, they had to verify minor yet critical details, such as ensuring the even probability distribution for the random number in the range of 1 to 100 inclusive.

The second module involved the loop controlling the 7 guessing attempts, which introduced several nuanced constraints. The program had to state the remaining number of guesses when prompting the user for a guess. Furthermore, it needed to suppress the "larger/smaller" hint on the final incorrect guess, replacing it with a definitive "You lose" message alongside the correct target number. Conversely, a correct guess should get the "You win!" response before program termination. Finally, students had to enforce optimal scope constraints and eliminate code duplication.

The students attempting to have AI generate the complete programming assignment from a single AI prompt would invariably find that the raw code failed to meet all the explicit requirements. Consequently, they would be forced to engage in iterative prompt engineering with the AI tool and manual refactoring of the generated code. The workflow would shift the students' attitude; they became active editors of AI output in better engagement with code readability and modular integration. That served our purpose of using AI to help them learn programming, with the added benefit of experience in using AI.

5. IN-CLASS QUIZZES

To ensure objective, direct assessment of individual student comprehension, in-class quizzes accounted for 35% of the final course grade. We had 4 cumulative quizzes throughout the semester, focusing heavily on the material covered in the preceding quarter of the semester. Each quiz contained between 20 and 40 questions, comprising multiple-choice, true/false, and fill-in-the-blank types. Students were given a limited time duration to complete the quizzes under open-book and open-note policies, but AI assistance is prohibited.

We leveraged generative AI to generate short, syntactically dense Java code snippets designed to test code literacy and logical evaluation. The students would need to read and understand the code to answer correctly. While AI helped in the process of question generation, the questions needed manual correction and adjustments before deployment. Below are 4 representative examples of the quiz questions used.

4.1 Nested loops with Flow Control
The students were required to read and understand the Java code segment and determine the precise console output.

```
for (int i=1; i<=2; i++)
{
  for (int j=1; j<=3; j++)
  {
    if (j==2) break;
    system.out.print(i+" "+j+" ");
  }
}
```

- **Correct Answer:** 1 1 2 1
- **Pedagogical Intent:** The question assesses whether the student understands that the loop construct and that the break statement immediately terminates only the innermost loop layer,

ISSN: 2473-4901
v12 n6818

resetting execution back to the outer loop context.

4.2 Method Overloading and Compilation Resolution

The question presents a scenario where a method named mini(...) is called, supposing to determine the minimum of two numerical values in the arguments, a primitive integer and a floating point double.

```
int a = 22;
double b = 33.33;
double c = mini(a,b); // call mini(...)
```

The students had to identify which one of the following overloaded method declarations would be called by the compiler for execution.

```
(A) public static int mini(int a, int b) { ... }
(B) public static double mini(double a, double b) { ... }
(C) public static int mini(int a, double b) { ... }
(D) public static double mini(double a, int b) { ... }
```

- **Correct Answer:** (C)
- **Pedagogical Intent:** The question tests the student's understanding of compile-time polymorphism. The Java compiler resolves overloaded methods by matching the sequence of the parameter types (the method signature). The method's return type is not part of the signature. The correct answer (C) provides the exact match.

4.3 Class Member Access and Scope

This question tests the student's understanding of encapsulation and object-oriented scope restrictions within a class structure.

```
class Thing
{
  public int a; // public instance variable
  private int b; // private instance variable
  public static int c; // public static variable
  private static int d; // private static variable
  public static void myStaticMeth() // public static method
  { ... } // implementation here
}
```

Which one of the following description about the access privilege of the static method myStaticMeth() is correct?

```
(A) myStaticMeth() can access instance member variables a and b only.
(B) myStaticMeth() can access public member variables a and c only.
(C) myStaticMeth() can access static member variables c and d only.
(D) myStaticMeth() can access all four member variables a,b,c and d.
```

- **Correct Answer:** (C)
- **Pedagogical Intent:** Since static methods belong to the class template rather than an instantiated object, they cannot implicitly reference instance variables (a or b) without an explicit object reference. They retain access to all static variables (c and d), regardless of their privacy modifiers.

4.4 Exception handling

The students must track the iterations across a loop construct, which we assumed they are reasonably familiar with. They then had to analyze exception handling blocks across the iterations to determine the console output.

```
for (int i=1; i<=2; i++) // loop runs 2 times
{
   try {
     System.out.print(" 1");
     // throw exception only on 1st run...
```

ISSN: 2473-4901
v12 n6818

```
      if (i==1) throw (new Exception("Null"));
    }
    catch (Exception e) {
      System.out.print(" 2");
      continue;
    }
     System.out.print(" 3");
}
System.out.println(" 4");
```

- **Correct Answer:** 1 2 1 3 4
- **Pedagogical Intent:** This requires tracking loop execution across iterations. The first iteration (i=1) console outputs " 1". The exception is thrown, execution skips to the catch block in which console outputs " 2". The continue statement ends the iteration to start the second iteration (i=2). Console outputs " 1". No exception is thrown. The catch block is skipped. Console outputs " 3". Upon exiting the loop, Console outputs " 4". Hence the correct answer is 1 2 1 3 4. The question assumes comprehension of the loop construct and tests the student's understanding of exception handling and the use of throw(Exception) method.

AI was a great help for us to generate quiz questions for use, even when it required detailed work to review, edit and verify. Since the students could not use AI in the in-class quizzes, they became a good direct assessment of the students' understanding of Java programming features.

6. ASSESSMENT

The two grading components in the course were the programming assignments and the in-class quizzes. They provided direct assessment of performance scores in grades. We also administered an anonymous post-course survey as indirect assessment of the students' perception of AI utility. The course enrolled 12 undergraduate students, but one student withdrew toward the end of the semester for family issues.

5.1 Direct Assessment

We encouraged and guided the students to use AI to help with their work on the programming assignments. Our hope was not only that the students get the experience of using AI, but also that they became interested in reading the code generated by AI. Despite rigorous grading constraints on code comments and design practices, the class average for the 10 programming assignments reached 73%. For the four invigilated, in-class quizzes in which the students could not use AI, the class average reached 83%. This strong performance on conceptual examinations indicated that AI integration during the programming assignments did not serve as a crutch; rather, it reinforced their independent comprehension. The final grade distribution was more encouraging: 3 students earned an A, 4 achieved an A-, 2 received a B+, and 2 finished with a B.

5.2 Indirect Assessment (Student Survey)

Out of 11 students who completed the course, 9 filed the anonymous end-of-semester survey evaluating their relationship with AI tools. The tabulated results are presented in Table 3 below.

It is understandable that the students were only using ChatGPT and Gemini since these were the AI tools we guided the students to use. The data suggests that our restrictive criteria successfully disincentivized wholesale code generation (only one student reported using AI to complete assignments in full). Instead, students mostly used AI as an interactive aid and an on-demand tutor. The lower mean score for software design (2.88) aligns with our observation that while generative AI is proficient at generating code segments to meet specifications, it struggles with holistic system integration.

Half of the cohort voiced concerns that AI could reduce their drive to learn. We need to investigate further about their concerns. However, their strong quiz scores suggest otherwise since the quizzes are done with no AI assistance. Additionally, the sentiment that AI use feels like plagiarism indicates an ethical transition period; students simply require explicit pedagogical frameworks to understand that reading, evaluating, and adapting public or generated code is an authentic, industry-standard programming practice.

(1)	**AI Tools used** (select all that apply)		
	(1.1)	ChatGPT	4 students
	(1.2)	Gemini	5 students
(2)	**Functions of AI use** (select all that apply)		
	(2.1)	Completion of programming assignments	1 student
	(2.2)	Answers to specific programming questions	5 students
	(2.3)	Assistance to debug programs	6 students
(3)	**How helpful was AI in these functions** Scale: 4=very helpful, 3=somewhat helpful, 2=barely helpful, 1=not helpful		
	(3.1)	To understand Java programming	Mean: 3.33
	(3.2)	To learn features in Java programming	Mean: 3.44
	(3.3)	To find bugs in programs	Mean: 3.33
	(3.4)	To learn program design	Mean: 2.88
(4)	**Concerns in using AI for help**		
	(4.1)	Hinders with desire to learn	5 students
	(4.2)	Feels like plagiarism	4 students
	(4.3)	Breaches my privacy	1 student

TABLE 3. Student Survey Results

7. SUMMARY AND DISCUSSION

AI has proliferated across the information technology and software engineering landscapes. Consequently, Information Systems educators must move past defensive avoidance and proactively integrate these tools into introductory programming courses. This paper reported an exploratory case study to leverage AI to maximize code literacy through structured code editing and integration.

By pairing AI-assisted programming assignments with strict documentation requirements and invigilated code-comprehension quizzes, we achieve a reasonable equilibrium. Direct and indirect assessment metrics confirmed that the students cultivated strong programming competencies while gaining realistic experience with prompt engineering and code review. In our exploratory study, the students' concern that AI hinders with their desire to learn and that it feels like plagiarism will need further investigation. Since the direct assessment results seem to indicate otherwise, our feeling is that the students may need to get used to the new learning environment with AI assistance.

As AI continues to evolve, our instructional approaches must similarly adapt: shifting the focus of introductory coursework away from syntax memorization and toward code literacy, architectural composition, and systems integration.

8. REFERENCES

Avouris, N., Sgarbas, K., Caridakis, G. & Sintoris, C. (2025). Teaching Introduction to Programming in the times of AI: A case study of a course re-design. *Proceedings 12th Penhellenic Conference of Computer Science Education*. PCCSE 2025, Rhodes, October 2025. https://arxiv.org/pdf/2508.06572

Azoulay, R., Hirst, T. & Reches, S. (2025). Large Language Models in Computer Science Classrooms: Ethical Challenges and Strategic Solutions. *Applied Science 2025, 15 (4), 1793*. https://www.mdpi.com/2076-3417/15/4/1793

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveria Pinto, H.P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N. et al. (2022). Evaluating Large Language Models Trained on Code, *arXiv, 2021*. http://arxiv.org/abs/2107.03374.

Finnie-Ansley, J., Denny, P., Becker, B.A., Luxton-Reilly, A. & Prather, J. (2022). The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. Australasian Computing Education Conference. *(ACE'22) Virtual Event, Australia.* ACM, New York NY, USA. https://doi.org/10.1145/3511861.3511863.

Frydenberg, M. (2026). From Excel to Python to AI: A Connectivist Model for Introducing Coding and Prompt Engineering to First-Year IS Students. *Information Systems Education Journal 24(2),27-43*. https://isedj.org/2026-24/n2/ISEDJv24n2p27.html

Garzon, J., Patino, E. & Marulanda, C. (2025). Systematic Review of Artificial Intelligence in Education: Trends, Benefits, and Challenges. Multimodal Technol. Interact 9(8), 84. https://doi.org/10.3390/mti9080084

Gosling, J., Joy, B., Steele, G. Buckley, A., Smith, D. & Bierman, G. (2025). The Java Language Specification (Java SE 25 Edition). Oracle America, Inc. https://docs.oracle.com/javase/specs/jls/se25/jls25.pdf

Hadi, M.A., Abdulredha, M.N. & Hasan, E. (2023). Introduction to ChatGPT: A new revolution of artificial intelligence with machine learning algorithms and cybersecurity. *Science Archives, 4(4), 276-285*. https://doi.org/10.47587/SA.2023.4406

Imran, M. & Almusharraf, N. (2024). Google Gemini as a next generation AI educational tool: a review of emerging educational technology. Smart Learning Environments 2024(11:22). https://doi.org/10.1186/s40561-024-00310-z

Kernighan, B.W. & Ritchie, D.M. (1978). The C Programming Language. Englewood Cliff: Prentice Hall. ISBN: 978-013-110163-0

MacNeil, S., Kim, J., Leinonen, J., Denny, P., Bernstein, S., Becker, B.A., Wermelinger, M., Hellas, A., Tran, A., Sarsa, S., Prather, J. & Kumar, V. (2023). The Implications of Large Language Models for CS Teachers and Students, P*roceedings of the 54th ACM Technical Symposium on Computer Science Education V.2*, March 2023, Toronto, Ontario, Canada. https://dl.acm.org/doi/10.1145/3545947.3573358

Marron, M. (2024). A New Generation of Intelligent Development Environments. First IDE Workshop (IDE '24), Lisbon, Portugal. *ACM, New York, NY, USA.* https://doi.org/10.1145/3643796.3648452.

Mayer, R.E. (1981). The psychology of how novices learn computer programming. *ACM Computing Surveys 13(1), 121-141*. https://doi.org/10.1145/356835.356841

Ng, D.T.K., Chan, E.K.C. & Lo, C.K. (2025). Opportunities, Challenges and School Strategies for Integrating Generative AI in Education. *Computers and Education: Artificial Intelligence 8, 100373*. https://doi.org/10.1016/j.caeai.2025.100373

Philbin, C.A. (2023). Exploring the Potential of Artificial Intelligence Program Generators in Computer Programming Education for Students. *ACM Inroads, 14(3), 30–38*. https://doi.org/10.1145/3610406

Prather, J., Denny, P., Leinonen, J., Smith, D. H., Reeves, B. N., MacNeil, S., Becker, B. A., Luxton-Reilly, A., Amarouche, T., & Kimmel, B. (2024). Interactions with Prompt Problems: A New Way to Teach Programming with Large Language Models. *arXiv preprint*. https://arxiv.org/abs/2401.10759

Raihan, N., Siddiq, M.L., Santos, J.C.S. & Zampieri, M. (2025). Large Language Models in Computer Science Education: A Systematic Literature Review. *Proceedings of the 56th ACM Technical Symposium on Computer Science Education. ACM, New York NY*, USA. https://arxiv.org/pdf/2410.16349

Sentance, S., Waite, J. & Kallia, M. (2019). Teaching computer programming with PRIMM: a sociocultural perspective. *Computer Science Education, 29(2-3), 136–176*. https://doi.org/10.1080/08993408.2019.1608781

Webb, M.E., Fluck, A., Magenheim, J., Malyn-Smith, J., Waters, J., Deschenes, M. & Zaqami, J. (2021). Machine learning for Human Learners: Opportunities, Issues, Tensions and Threats. *Education Tech Research Dev 69, 2109–2130*. https://doi.org/10.1007/s11423-020-09858-2

Zviel-Girshin, R. (2024). The Good and Bad of AI Tools in Novice Programming Education. *Education Sciences*, *14(10), 1089*. https://doi.org/10.3390/educsci14101089