Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Video Game Programming with Python to Develop Logical Thinking in High School: A Methodological Proposal to Counter Cognitive Overload in the Age of Artificial Intelligence.

Miguel Ángel Suárez-Ledo
Yachay Experimental Technology Research University
Yesenia Pamela Amaya-Guapisaca
Yachay Experimental Technology Research University
Sabina Caula
sabinacaula@gmail.com

Universidad Internacional De La Rioja

Research Article

Keywords: Logical thinking, Educational programming, Python, Cognitive offloading, Generative artificial intelligence, STEM education, Projectile motion

Posted Date: November 11th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10944131/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Video Game Programming with Python to Develop Logical Thinking in High School: A Methodological Proposal to Counter Cognitive Overload in the Age of Artificial Intelligence.

Abstract

The development of logical thinking is a core competency in STEM education that assessments identify as lacking in the High School Program. Compounding this gap is an emerging phenomenon: the tendency among students to rely on generative artificial intelligence (AI) tools to solve problems and submit answers without conducting their own analysis—a pattern of cognitive offloading that recent literature associates with the erosion of critical thinking and the loss of judgment. This article presents a methodological guide that uses video game programming with Python and the Tkinter library to strengthen logical reasoning in High School students, with the simulation of projectile motion serving as the central focus. The proposal is grounded in constructivism, Kolb's experiential learning theory, and active learning methodologies, and is organized as design-based research, reporting on the design and internal validation phase of a prototype. The instructional sequence progresses through sixteen modules that guide students from basic syntax to a cannon simulator with real-world physics. It is argued that positioning the student as a producer of logic—who breaks down problems, encodes the physical model, and debugs their own errors—constitutes a deliberate pedagogical counterbalance to cognitive offloading, because it restores the cognitive effort and productive challenge that the uncritical use of AI tends to eliminate. The expected learning outcomes, the formative assessment rubric, and the implications for educational policy are discussed.

Keywords Logical thinking · Educational programming · Python · Cognitive offloading · Generative artificial intelligence · STEM education · Projectile motion

Introduction

Education in science, technology, engineering, and mathematics (STEM) is underpinned by a cross-curricular competency: logical thinking, understood as the ability to break down complex problems, structure algorithmic sequences, abstract causal relationships, and rigorously validate conclusions (Jaramillo & Puga, 2016). At the regional level, international assessments, such as the PISA tests (OECD, 2023), place South America in a critical position, where performance in science and logical reasoning remains persistently below global standards. In Ecuador, this regional gap is evident and alarming: reports from the National Institute for Educational Assessment reveal structural deficiencies in logical reasoning in the areas of mathematics and physics within the High School Programs (Ineval, 2024). This deficit not only compromises academic success but also leads, in the medium term, to high dropout rates during the transition to higher education (SENECYT, 2021). These results highlight the limitations of a pedagogical model that, while prevalent in Latin American classrooms, still prioritizes the rote memorization of formulas and the reproduction of procedures over the active and reflective construction of knowledge (Quimis et al., 2022).

In recent years, this structural problem has been compounded by a rapidly expanding phenomenon that is altering the way students engage with cognitive tasks. The emergence of generative artificial intelligence (AI)—and in particular, large language models—has put within any student's reach a tool capable of producing, in seconds, the apparent solution to a physics problem, a snippet of code, or an entire essay (Kasneci et al., 2023). The pedagogical risk does not lie in the tool itself, but rather in a pattern of use that is being observed with increasing frequency: the student formulates the prompt for the AI, copies the response, and submits it without having gone through the reasoning process that the task was intended to elicit. This shift of cognitive effort toward an external agent is a form of cognitive offloading (Risko & Gilbert, 2016), and emerging evidence suggests that its systematic and uncritical use is associated with reduced engagement in critical thinking and with what has been termed metacognitive laziness (Fan et al., 2024; Gerlich, 2025).

The problem that motivates this study is, therefore, twofold and interrelated. On the one hand, there remains a historical deficit in the development of logical thinking in high school education.

On the other hand, the practice of uncritical reliance on AI threatens to deepen this deficit by depriving students of precisely the kind of effort—breaking down, formalizing, verifying, correcting—that builds their judgment. Given this scenario, pedagogical approaches are needed that go beyond merely incorporating technology and instead reposition students as producers of reasoning rather than consumers of answers.

Programming offers particularly fertile ground for this purpose. Writing a program requires explicitly articulating each step of the reasoning process, anticipating the system's behavior, and immediately confronting the consequences of one's own errors through debugging (Guzdial, 2015; Lodi & Martini, 2021). When the programming project is a video game that simulates a physical phenomenon, the activity also becomes motivating and meaningful, as it integrates physics and mathematics curriculum content with a creative and tangible practice (Vázquez-Cano & Ferrer, 2015; Denner et al., 2009). On this basis, this article describes a methodological guide that uses Python and the Tkinter graphics library to enable High School students to progressively build a projectile motion simulator, and explicitly positions this approach as a counterbalance to cognitive overload.

In light of this scenario, this research proposes a methodology based on video game programming with Python, designed with a focus on curricular versatility that allows for its cross-curricular integration into any high school program (Author, 2025). The study is guided by three research questions. First: What theoretical and methodological foundations support the use of video game programming with Python as a means to develop logical thinking at the high school level? Second: How can a replicable methodological guide be structured to lead students from basic syntax to the simulation of projectile motion? Third: How can an approach centered on the development of logical thinking counteract the cognitive overload and loss of judgment associated with the uncritical use of generative AI?

Theoretical Framework and Literature Review

This section outlines the four bodies of knowledge upon which the proposal is based: the nature of logical thinking and the learning theories that guide its development; the role of programming and the STEM approach as pathways to cultivate it; the physical model of projectile motion that

serves as the unifying content; and, most notably, the phenomenon of cognitive offloading in the era of generative AI, which gives meaning and urgency to the proposal.

Logical Thinking, Constructivism, and Experiential Learning

Logical thinking is understood here as the ability to organize reasoning into coherent sequences, establish cause-and-effect relationships, abstract common structures across different problems, and evaluate the validity of inferences (Aristóteles, 1987; Campistrous, 2007; Jaramillo & Puga, 2016). It is not a matter of content being transmitted, but rather a competency that is constructed through the student's own activity. This premise draws on constructivism, according to which knowledge is actively constructed based on experience and the reorganization of prior cognitive structures (Veliz, 2022).

Kolb's (1984) experiential learning model provides a practical framework for this process. His cycle describes how learning progresses from concrete experience to reflective observation, then to abstract conceptualization, and finally to active experimentation, which in turn generates a new concrete experience in an upward spiral. Video game programming fits naturally into this cycle: the student runs a piece of code (concrete experience), observes its behavior and errors (reflective observation), formulates the principle that explains that behavior (abstract conceptualization), and modifies the code to test their hypothesis (active experimentation).

Active methodologies, such as project-based learning and collaborative learning, provide the pedagogical framework that makes this journey possible by shifting the classroom's focus from the teacher's presentation to student-centered production (Quimis et al., 2022).

Programming, the STEM Approach, and Python in Secondary Education

The STEM approach promotes the integration of scientific and technological disciplines around real-world problems, rather than teaching them in isolation (Vázquez et al., 2013; López et al., 2020). Within this framework, programming serves a dual purpose: it is both a subject matter and a method. As a method, it requires students to formalize their reasoning in unambiguous language, which makes logical errors visible and correctable (Guzdial, 2015; Villacrés-Sampedro et al., 2024). Various studies have shown that creating video games and simulations is an effective way

to introduce these concepts in secondary education, due to their ability to combine technical rigor with motivation (Barrera & López, 2021; Vázquez-Cano & Ferrer, 2015).

Python has established itself as an ideal first language for this purpose. Its clear syntax, which closely resembles natural language, reduces the cognitive load associated with syntax, allowing students to focus their efforts on the logic of the problem (García-Monsalvez, 2017; Mészárosová, 2015; Duque, 2015). Designed from the outset with a focus on accessibility and "programming for everyone" (Van Rossum, 1999), Python also features the Tkinter library, included in its standard distribution, which allows users to build graphical interfaces and animations with a small number of instructions. This feature is crucial from an educational standpoint: it enables a student with no prior experience to create, in just a few sessions, a visual and interactive product that represents a physical phenomenon.

Projectile Motion as an Integrating Topic

Projectile motion is an ideal curricular topic for integrating physics, mathematics, and programming. It involves two-dimensional motion in which the horizontal component of acceleration is zero and the vertical component corresponds to the acceleration due to gravity (Young & Freedman, 2013). The projectile's position is described by the equations of uniform motion along the horizontal axis and uniformly accelerated motion along the vertical axis, while the decomposition of the initial velocity into its components requires the use of trigonometric ratios.

For the initial velocity, we have:

$$v_{0x}=v_0 \cdot cos(\theta)$$

$$v_{0y}=v_0 \cdot sin(\theta)$$

While for the position:

$$x=x_o+v_{ox}t$$

$$y=y_0+v_{0y}t-\frac{1}{2}gt^2$$

This content connects directly to the curriculum of any secondary education program: to quadratic functions and trigonometric ratios in mathematics, and to the study of projectile motion in physics. Translating these equations into code forces students to understand the physical model with a precision that mere algebraic manipulation does not always require, since the program only behaves correctly if the underlying reasoning is correct.

Generative AI, Cognitive Offloading, and Loss of Judgment

Cognitive offloading refers to the use of external resources—ranging from a note on paper to a search engine—to reduce the processing demands on one's own memory and reasoning systems (Risko & Gilbert, 2016). This is an ancient and, in many contexts, adaptive mechanism; the extended mind thesis even argues that a significant portion of human cognition legitimately relies on the environment (Clark & Chalmers, 1998). The problem arises when offloading replaces, rather than complements, the processes that education seeks to develop. Research on the so-called "Google effect" has already shown that the immediate availability of information online tends to make people remember where to find a piece of information rather than the information itself (Sparrow et al., 2011). Generative AI takes this phenomenon to a new level, because it does not merely locate information: it directly produces the finished product—the solution to the problem, the code, the text—and thereby offers the possibility of completely bypassing the reasoning process.

Recent evidence, though still emerging, is convergent and concerning. Studies involving knowledge workers and students document that greater reliance on AI tools is associated with a self-reported reduction in the cognitive effort devoted to tasks and with less critical thinking (Lee et al., 2025; Gerlich, 2025). In learning contexts, experimental research shows that the scaffolding provided by generative AI can improve the immediate output without improving—and may even impair—processes of self-regulation and transfer of learning, a pattern described as metacognitive laziness (Fan et al., 2024). Students and teachers recognize both the benefits of these tools and the specific risk of a dependency that undermines intellectual autonomy and the ability to judge the quality of a response (Chan & Hu, 2023; Darwin et al., 2023; Çela et al., 2024).

The appropriate pedagogical response is not prohibition, but rather the redesign of learning tasks so that cognitive effort once again becomes inevitable and visible (Gašević et al., 2023). The psychology of learning offers a relevant principle: processes that impose a desirable level of difficulty—which require effort, generate errors, and necessitate correction—produce more lasting and transferable learning than those that pave the way (Sweller et al., 2019). In the specific field of computer science education, the emergence of generative AI has reignited the debate on how to preserve genuine learning in programming when code can be generated automatically (Prather et al., 2023). The proposal described here follows this line of thinking: rather than asking for an answer that AI could provide, it challenges students to build and debug a functional model—an activity in which judgment cannot be delegated without the product ceasing to function.

Methodology

This paper presents the design phase of the instructional sequence with internal validation; the empirical evaluation phase in the classroom is proposed as a future line of research (Author, 2025). Consequently, what is presented here is the design of a methodological guide and its technical validation, not a study of its impact on learning.

When using project-based learning, the teacher assumes the role of guide and facilitator of the process, in line with the active methodologies that underpin the proposal.

Context and Target Audience

The proposal is aimed at students between the ages of twelve and seventeen in schools equipped with computer labs. No prior programming experience is assumed. The choice of video games as a learning tool is based on students' high level of familiarity with this medium: available data indicate that nearly 77% of young people play video games regularly (Fundación de Ayuda contra la Drogadicción, 2023), making the creation of a game a meaningful and motivating goal.

Learning Environment and Tools

The technical environment consists of free and open-source software to ensure the proposal can be replicated in different schools. Python 3.x (Python Software Foundation, 2024), managed through the Anaconda distribution (Anaconda Inc., 2024), was used to develop the proposal. The integrated development environment (IDE) selected was Spyder (Spyder Project Contributors, 2024), using the Tkinter graphics library for the interface and the physics simulation.

The instructional sequence is designed to progress through three environments of increasing complexity that align with the student's cognitive development: first, the command console (PowerShell), to become familiar with executing basic instructions; next, the Spyder editor, to write and organize structured programs; and finally, Tkinter, to build the simulator's graphical interface and animations. Multimedia files (JPG images and WAV sounds) are incorporated as supplementary resources to enrich the final product.

Design of the Instructional Sequence

The guide is structured into sixteen progressive modules (Table 1), organized around an integrative project: the construction of a cannon simulator that represents the movement of projectiles (Author, 2025). In this system, the user assumes the role of a gunner with the objective of hitting a target located at specific coordinates (Fig. 1).

The first modules cover the installation and configuration of the environment, data types, operators, variables, and screen output statements. The intermediate modules address control structures—conditional statements and loops—and functions, which introduce the decomposition of a problem into subproblems. The final modules develop the graphical interface using Tkinter: creating the window and the canvas, drawing and animating objects, incorporating user input, and, as a culmination, implementing the physical model of the projectile with parameters entered by the student.

This progression embodies Kolb's learning cycle, as each module begins with a specific coding experience and progresses toward conceptualizing the underlying principle.

The integrative project is developed in three versions of increasing complexity. In the first, the student animates an object with simplified motion, using a reduced gravity value to observe

parabolic behavior without the burden of the full equations. In the second, the actual physical model is introduced, with a discrete time step (dt) and position updates based on the velocity components. In the third version, the program incorporates user input—the initial velocity and launch angle—and calculates the velocity components using the mathematical functions library, converting degrees to radians and applying sine and cosine functions. It is in this activity that all the learning comes together: the student must understand the physical model, translate it into code, handle user input, and debug errors until the simulated trajectory behaves in a physically correct manner.

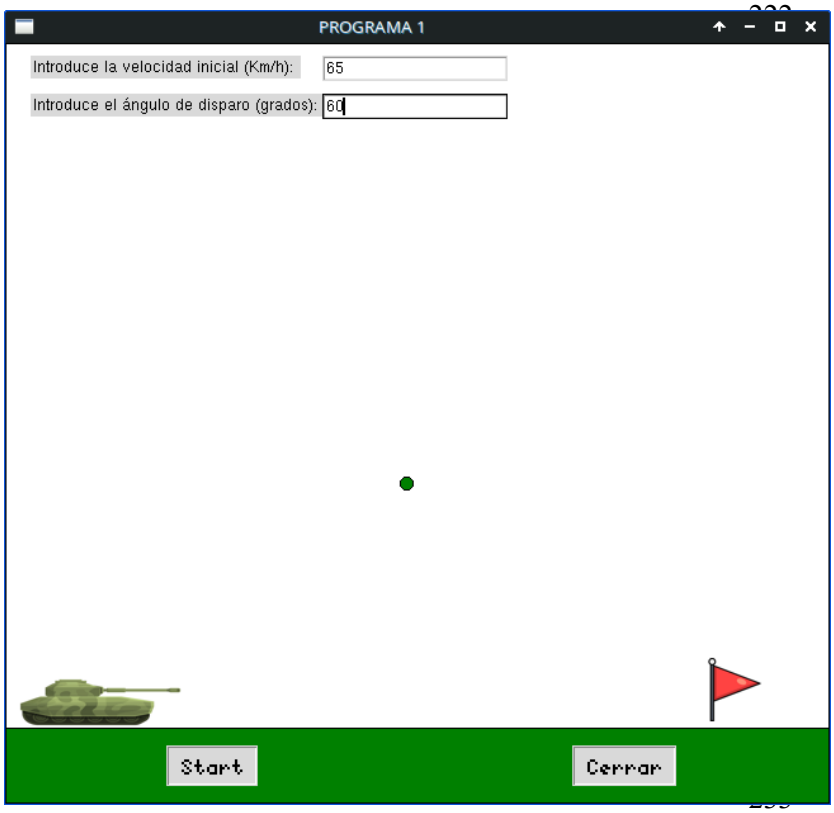


Figure 1. *Screenshot of the video game.*

Validation

Internal validation of the prototype was performed iteratively through successive software testing, debugging, and verification of the accuracy of the physical calculations by comparing the simulated trajectory with the expected behavior according to the equations of projectile motion

Table 1. Instructional sequence for teaching basic programming and the development of graphical user interfaces in Python to students in the Unified General High School program.

Blocks	Objectives / Learning Outcomes	Sessions
Initial exploration with the Python console	Introduce students to the interactive programming and execution environment. Understand the basic syntax of Python. Execute flow control structures. Gain a basic understanding of data input and output.	8
Transition to the integrated development environment (Spyder)	Compare interactive execution with programming using reusable scripts. Identify an integrated development environment for creating, developing, and saving *.py* files. Understand basic concepts such as importing and using Python libraries, basic error handling, and *function* concepts.	15
Introduction to the Standard Python GUI 1. Designing a graphical user interface with Tkinter 2. Physics Simulation and Animation 3. Multimedia and Immersive Experiences	Running Custom Graphical Windows Inserting basic widgets. Manage user events using commands ( ). Create geometric shapes using Canvas. Introduce concepts of kinematics using visual programming. Implement equations of parabolic motion in code. Control variables and update values visible in the console using methods. Quantify trajectories to achieve continuous animation.	20

	Integrate visual and sound effects. Manage external resources such as images or audio files. Adapt the graphical interface to resizing.	
Final Production	Consolidate the complete program into a functional and debugged simulator	1
	Total	44

Assessment

To assess learning in the classroom, the guide proposes formative and continuous assessment, supported by a rubric that identifies four levels of achievement based on five criteria: algorithmic logic, debugging, creativity, application of physics, and collaborative work (Table 2). This assessment shifts the focus from the final product to the reasoning process, which is consistent with the goal of counteracting the submission of results without independent analysis.

Table 2. Final Assessment Rubric

Criterion	Level 4	Level 3	Level 2	Level 1
Algorithmic logic	Optimized syntax. Original solutions.	Functional control structures. Minimal redundancy.	Repetitive code. Frequent logic errors.	Repeating code without understanding it.
Debugging	Detects and resolves advanced errors independently.	Fixes errors with minimal support.	Requires constant guidance to resolve errors.	Fails to identify obvious errors.
Creativity	Incorporates original mechanics. Suggests alternative designs.	Alter predefined values.	Applies rules without variation.	Resists innovation.

Applied Physics	Adjusts 3 or more variables with mathematical rigor.	Properly controls two variables. Minor calculation errors.	Uses a variable inflexibly. Generates inconsistent trajectories.	Does not relate physics to code.
Collaborative work	Explains concepts to peers. Leads team activities.	Actively participates in discussions. Exchanges ideas.	Limited contribution to group work.	No interaction with peers.

Discussion and Conclusions

This sequence is designed to strengthen a specific set of logical thinking skills. Breaking down the simulator into subproblems—initializing variables, calculating components, updating the position, drawing the object—exercises abstraction and sequential analysis. Writing control structures and functions develops the ability to formalize reasoning in unambiguous steps. Debugging, a central component of the activity, trains critical evaluation: students must interpret the program's behavior, formulate hypotheses about the cause of the error, and verify them—a cycle of reasoning that the rubric explicitly recognizes. Finally, translating the kinematic equations into code reinforces the application of physics and trigonometry in a meaningful context. Added to this is a motivational benefit: students cease to be mere consumers of technology and become its creators—a role shift that the literature associates with greater engagement in the task (Denner et al., 2009; Vázquez-Cano & Ferrer, 2015).

The design presented addresses the three research questions that guided this study. Regarding the first question, the proposal is grounded in a coherent theoretical framework that links constructivism, Kolb's experiential cycle, and active methodologies with the potential of programming as a means to make reasoning explicit and to practice it. Regarding the second question, the guide offers a replicable sequence of sixteen modules—aligned with the secondary education curriculum—that gradually progresses from basic syntax to a projectile simulator with real-world physics, and whose internal validation confirms its technical feasibility.

The third question—how can a proposal of this kind counteract the cognitive offloading and loss of judgment associated with the uncritical use of generative AI—deserves a more detailed discussion, as it constitutes this article's contribution to the previous line of research. The central argument is that the activity of building and debugging a simulator, by its very nature, reintroduces the cognitive effort that uncritical reliance on AI tends to suppress. When a student asks a language model for the answer to a physics problem, they may obtain a correct result without having reasoned through it; the output masks the absence of a thought process. In contrast, a malfunctioning simulator cannot be concealed: the erroneous trajectory on the screen provides immediate and indisputable feedback that forces the student to revisit their reasoning. Debugging, in this sense, is the pedagogical opposite of cognitive offloading, because it does not allow a result to be delivered without analysis.

This does not mean that the proposal excludes AI, nor that it should. Generative AI can play a legitimate role as an assistant—to explain an error message or suggest a design alternative—provided that the task is structured so that understanding and judgment remain the student's responsibility (Gašević et al., 2023). The key lies in the task design: by requiring a functional product whose correctness depends on the internal coherence of the reasoning, the activity makes cognitive effort an unavoidable condition and exposes any attempt to bypass it. It is, in the terms of the psychology of learning, a desirable difficulty deliberately incorporated into the design (Sweller et al., 2019). In this way, the proposal not only develops logical thinking as a curricular objective but also cultivates the willingness to sustain intellectual effort—the discernment—that the culture of instant gratification threatens to erode.

The study has limitations that should be made explicit. Since only the design and internal validation phase is reported, empirical evidence is not yet available regarding the guide's impact on students' actual learning or its specific effect on AI usage patterns. Verification of the pedagogical hypotheses formulated here requires a classroom implementation phase, using instruments that measure both the development of logical thinking and self-regulation, as well as reliance on external tools. This empirical evaluation constitutes the main line of future work, along with the expansion of the proposal to other topics in physics and mathematics.

In conclusion, video game programming with Python emerges as a teaching tool—not an end in itself—for democratizing the development of logical thinking in high school. Its value is amplified in the current context: in the face of the tendency to delegate reasoning to generative AI, a task that requires building, verifying, and correcting a functional model restores the student's role as a producer of logic and safeguards the exercise of critical judgment. The barriers to its adoption are primarily cultural and related to educational policy rather than technical, so its widespread adoption requires the support of educational institutions and authorities.

Disclosures

Funding: The authors did not receive specific funding for this research.

Conflict of Interest: The authors declare that they have no conflicts of interest.

Data Availability: The source code for the prototype is available upon reasonable request to the authors.

References

Author (2025)

Anaconda Inc. (2024). *Anaconda Distribution* (Versión software). https://www.anaconda.com/

Aristóteles. (1987). *Tratado de lógica (Órganon)*. Porrúa.

Barrera, C., & López, A. (2021). Scratch: La programación como detonante del pensamiento matemático. *Dialnet*, Educando para educar 22(41), 119-130.

Campistrous, L. A. (2007). *El proceso de enseñanza-aprendizaje del desarrollo en la escuela secundaria inferior*. La Habana: Centro de Estudios Educacionales.

Çela, E., Fonkam, M., & Potluri, R. M. (2024). Risks of AI-assisted learning on student critical thinking. *International Journal of Risk and Contingency Management*, 12(1), 1–19. https://doi.org/10.4018/ijrcm.350185

Chan, C. K. Y., & Hu, W. (2023). Students' voices on generative AI: Perceptions, benefits, and challenges in higher education. *International Journal of Educational Technology in Higher Education*, 20(1), Artículo 43. https://doi.org/10.1186/s41239-023-00411-8

Clark, A., & Chalmers, D. J. (1998). The extended mind. *Analysis*, 58(1), 7–19. https://doi.org/10.1093/analys/58.1.7

Darwin, Rusdin, D., Mukminatien, N., Suryati, N., Laksmi, E. D., & Marzuki. (2023). Critical thinking in the AI era: An exploration of EFL students' perceptions, benefits, and limitations. *Cogent Education*, 11(1), Artículo 2290342. https://doi.org/10.1080/2331186X.2023.2290342

Denner, J., Bean, S., & Martinez, J. (2009). The Girl Game Company: Engaging Latina girls in information technology. *Afterschool Matters*, 8, 26–35.

Duque, R. (2015). *Python para todos*. Creative Commons.

Fan, Y., Tang, L., Le, H., Shen, K., Tan, S., Zhao, Y., Shen, Y., Li, X., & Gašević, D. (2024). Beware of metacognitive laziness: Effects of generative artificial intelligence on learning motivation, processes, and performance. *British Journal of Educational Technology*, 56(2), 489–530. https://doi.org/10.1111/bjet.13544

Fundación ayuda contra la Drogadicción [FAD]. (2023). Video juegos y jóvenes: lugares, experiencias y tensiones. FAD. 2023.Videojuegos-y-jovenes.Informe-final.pdf

García-Monsalvez, J. (2017). Python como primer lenguaje de programación basado en texto en educación secundaria. *Education in the Knowledge Society*, 18(2), 147–162.

Gašević, D., Siemens, G., & Sadiq, S. (2023). Empowering learners for the age of artificial intelligence. *Computers and Education: Artificial Intelligence*, 4, Artículo 100130. https://doi.org/10.1016/j.caeai.2023.100130

Gerlich, M. (2025). AI tools in society: Impacts on cognitive offloading and the future of critical thinking. *Societies*, 15(1), Artículo 6. https://doi.org/10.3390/soc15010006

Guzdial, M. (2015). *Learner-centered design of computing education: Research on computing for everyone*. Synthesis Lectures on Human-Centered Informatics, 8(6), 1–165.

Instituto Nacional de Evaluación Educativa [Ineval]. (2024). *Políticas transformadoras: hacia el nuevo Ecuador, desde la evaluación educativa*. Ineval.

Jaramillo, L., & Puga, L. (2016). El pensamiento lógico-abstracto como sustento para potenciar los procesos cognitivos en la educación. *Sophia, Colección de Filosofía de la Educación*, (21), 31–55.

Kasneci, E., Seßler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., Krusche, S., Kutyniok, G., Michaeli, T., Nerdel, C., Pfeffer, J., Poquet, O., Sailer, M., Schmidt, A., Seidel, T., & Kasneci, G. (2023). ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103, Artículo 102274. https://doi.org/10.1016/j.lindif.2023.102274

Kolb, D. A. (1984). *Experiential learning: Experience as the source of learning and development*. Prentice Hall.

Lee, H.-P., Sarkar, A., Tankelevitch, L., Drosos, I., Rintel, S., Banks, R., & Wilson, N. (2025). The impact of generative AI on critical thinking: Self-reported reductions in cognitive effort and confidence effects from a survey of knowledge workers. En *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems* (pp. 1–22). ACM. https://doi.org/10.1145/3706598.3713778

Lodi, M., & Martini, S. (2021). Computational thinking, between Papert and Wing. *Science & Education*, 30(4), 883–908. https://doi.org/10.1007/s11191-021-00202-5

López, V., Couso, D., & Simarro, C. (2020). Educación STEM en y para un mundo digital: El papel de las herramientas digitales en las prácticas científicas, de ingeniería y matemáticas. *REDU. Revista de Docencia Universitaria*, 62(20).

Mészárosová, E. (2015). Is Python an appropriate programming language for teaching programming in secondary schools? *International Journal of Information and Communication Technologies in Education*, 4(2), 5–14. https://doi.org/10.1515/ijicte-2015-0005

Ministerio de Educación de Ecuador. (2019). *Currículo de los niveles de educación obligatoria: Bachillerato General Unificado*. Ministerio de Educación.

OECD. (2023). *PISA 2022 Results (Volume I): The state of learning and equity in education*. OECD Publishing. https://doi.org/10.1787/53f23881-en

Prather, J., Denny, P., Leinonen, J., Becker, B. A., Albluwi, I., Craig, M., Keuning, H., Kiesler, N., Kohn, T., Luxton-Reilly, A., MacNeil, S., Petersen, A., Pettit, R., Reeves, B. N., & Šavelka, J. (2023). The robots are here: Navigating the generative AI revolution in computing education. En *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education* (pp. 108–159). ACM. https://doi.org/10.1145/3623762.3633499

Python Software Foundation. (2024). *Python 3.12.x* (Versión software). https://www.python.org/

Quimis, C., García, G., & Saltos, R. (2022). Los métodos didácticos en el desarrollo del pensamiento lógico . *Cognosis Revista*, 7, 77–92.

Risko, E. F., & Gilbert, S. J. (2016). Cognitive offloading. *Trends in Cognitive Sciences*, 20(9), 676–688. https://doi.org/10.1016/j.tics.2016.07.002

Secretaría de Educación Superior, Ciencia, Tecnología e Innovación [SENESCYT]. (2021). *Sistema de nivelación y admisión*. SENESCYT.

Sparrow, B., Liu, J., & Wegner, D. M. (2011). Google's effects on memory: Cognitive consequences of having information at our fingertips. *Science*, 333(6043), 776–778. https://doi.org/10.1126/science.1207745

Spyder Project Contributors. (2024). *Spyder IDE* (Versión software). https://www.spyder-ide.org/

Sweller, J., van Merriënboer, J. J. G., & Paas, F. (2019). Cognitive architecture and instructional design: 20 years later. *Educational Psychology Review*, 31(2), 261–292. https://doi.org/10.1007/s10648-019-09465-5

Van Rossum, G. (1999). *Computer programming for everybody: A scouting expedition for the programmers of tomorrow*. Python Software Foundation.

Vázquez-Cano, E., & Ferrer, D. (2015). Creación de videojuegos con Scratch en educación secundaria. *Communication Papers*, 4(6), 63–73.

Vázquez, J. A., Sneider, C., & Comer, M. (2013). *STEM lesson essentials, grades 3–8: Integrating science, technology, engineering, and mathematics*. Heinemann.

Veliz, C. (2022). *Fundamentos del enfoque constructivista en la atención educativa de niños de tres años* [Tesis de licenciatura, Pontificia Universidad Católica del Perú]. Repositorio PUCP.

Villacrés-Sampedro, J., Cordovez-Machado, S., Montoya-Ramírez, N., & Calvopiña-Estrella, F. (2024). Fundamentos de programación: Un acercamiento al desarrollo del pensamiento. *Polo del Conocimiento*, 9(3), 3280–3299.

Young, H. D., & Freedman, R. A. (2013). *University Physics* (Vol. 1, 13.ª ed.). Pearson.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- Supplementarymaterial.pdf