LEARNING BY PROMPTING AS AN EXTENSION OF LEARNING BY MAKING IN TURTLE GEOMETRY PROGRAMMING

Mario Lepore[1], Roberto Capone[2], Eleonora Faggiano[2]

[1]University of Salerno, [2]University of Bari Aldo Moro

Generative AI is increasingly entering mathematics classrooms, yet it may shift students' work toward uncritical acceptance of generated outputs. To counter this risk, we analyse how learning by prompting can extend constructionist learning by making, supporting mathematical thinking within programming activities. We report a classroom-based qualitative case study in an Italian scientific high school where small groups worked on a Python Turtle geometry task with a customised GenAI assistant. We trace how prompts evolved from vague requests to explicit mathematical specifications across prompt–debug–refine cycles. Results show that mismatches between intended and produced outputs triggered revisions that made constraints more explicit and supported decomposition of the geometric structure.

INTRODUCTION AND RATIONALE

Generative Artificial Intelligence (GenAI), and in particular Large Language Models, is rapidly entering mathematics classrooms. In mathematics education, these tools are discussed as sources of explanations and examples, as conversational partners, and as supports for students' work (Pepin et al., 2025). Alongside potential benefits, recent work highlights recurring risks: students may accept generated outputs without checking them, mathematical errors may pass unnoticed, and classroom activity may shift from making sense to "getting something that works" (Walkington, 2025). For this reason, a key research need is not only to ask whether GenAI is useful, but to understand how it can support students' mathematical thinking through observable practices such as specifying constraints, testing outputs, and revising solutions during classroom activity. In this paper we focus on a specific possibility: using GenAI to strengthen, rather than replace, students' role in expressing, checking, and validating mathematical ideas while they work in a programming environment. Our starting point is constructionism. In Papert's view, learning is supported when students build shareable objects and improve them through cycles of trying, noticing what does not fit, and revising (Papert, 1980). With GenAI, this cycle can take a new form: students may work not only by editing code, but also by revising the written specifications they give to the system. In particular, learning by prompting can be seen as an extension of learning by making, where students gradually turn a vague intention into a more precise description that a system can execute and that a group can inspect, test, and correct. In this sense, prompts can function as evolving specifications: they make constraints explicit (e.g., symmetry, angles, repetition rules, segment lengths, headings, pen

2026. In H. Krzywacki, M. Hähkiöniemi, E. Haataja, M.S. Hannula & P. Hästö (Eds.). *Proceedings of the 49th Conference of the International Group for the Psychology of Mathematics Education: Research Reports*, pp. 939-946. PME.

control), and they allow students to compare intention and output in a concrete way. Within this perspective, Turtle geometry (drawing by programming a cursor that moves and turns) is especially suitable because it provides immediate visual feedback and invites students to express geometric ideas through parameters, turns, and repeated patterns (Abelson, H., & diSessa, 1986). Our interest is therefore in how students' prompts change over time while they work, and in what this reveals about mathematical thinking in programming contexts supported by GenAI. Accordingly, we address the following research questions: RQ) *How do students' prompts evolve into explicit mathematical specifications across prompt–debug–refine cycles when working with a customised GenAI assistant in a Turtle-geometry programming task?* SubRQ: *What does this reveal about their mathematical reasoning?*

To explore these questions, we report a classroom-based qualitative case study conducted in an Italian scientific high school. Students worked in small groups on a Python Turtle task and interacted with a customised GenAI assistant while developing their solutions. The dataset includes interaction logs, students' worksheets, and the produced code and outputs. The study does not aim at generalisation, but at providing a detailed account of the mechanisms through which learning by prompting can keep mathematical reasoning at the centre of programming activity.

THEORETICAL FRAMEWORK

In this paper we read students' work with GenAI through a constructionist perspective. Our starting point is that, in a Turtle-based geometry task, GenAI can be used not only to produce code, but to support learning by prompting, as students translate an initial idea into progressively clearer mathematical specifications. To make this reading concrete, we connect constructionism to two related lenses: Problem-Based Learning (PBL) (Hmelo-Silver & Barrows, 2006), to characterise the didactical conditions of the activity, and Computational Thinking (CT) (Wing, 2006), to describe the practices that become visible during prompt–debug–refine cycles. We draw on constructionism (Papert, 1980), where learning is supported when students build shareable objects that can be inspected, discussed, and revised, turning the object into an "object to think with". In such settings, debugging plays a central role: mismatches between intention and outcome invite analysis, hypothesis-building, and revision. In our context, this mechanism applies not only to the Turtle program and its output, but also to the prompt, which functions as an evolving specification that students refine to make constraints explicit (e.g., symmetry, angles, repetition rules, segment lengths, headings, pen control). We use PBL as a descriptive lens for the task design and operationalise it through five features: (i) a meaningful goal that requires interpretation and choices; (ii) students' responsibility for exploring and revising attempts; (iii) collaborative negotiation; (iv) iterative testing and improvement; and (v) reflection on product and process. This helps us read prompting as part of an inquiry cycle rather than a shortcut to a ready-made output. We then adopt CT as a lens for describing practices that support mathematical work in this setting, operationalised through decomposition,

abstraction/parameterisation, algorithmic specification, and debugging. These categories allow us to trace how students move from broad requests to more precise mathematical descriptions, and how prompt refinement supports mathematical reasoning while programming. The following case study illustrates how this framework can be used to analyse students' work on a Python Turtle geometry task.

METHODS

To address our research questions, we designed and implemented a classroom-based activity in which students worked on a Turtle geometry programming task supported by a customised GenAI assistant, within an artifact-centred iterative design setting informed by constructionism. The activity was carried out with 17 students (9 male, 8 female) from a 13th-grade class at the Italian scientific high school "P. S. Mancini" in Avellino (Italy). Students worked in four small groups (three groups of four and one group of five). The session was introduced and facilitated by the authors, who presented the aims of the activity and the role of the tools and provided procedural support during the work. Students were provided with a booklet containing the task instructions and a worksheet to document their work.

Description of the designed activity

The activity was designed as a group-based design task in which students built a geometric artefact using Python Turtle and used a customised ChatGPT tutor as conversational support during the design and debugging process. The customised tutor was configured to interact as a "Logo/Turtle assistant": it encouraged students to state their geometric intention (e.g., shape, symmetries, angles, repetitions) before proposing code, to proceed in small steps, and to revise the solution through questions and incremental modifications rather than delivering a finished product immediately. We made this choice because GenAI systems are flexible: through a carefully designed system prompt, they can be adapted to act as a pedagogical support aligned with a specific classroom context (Bernardi et al., 2025). In addition, following Kynigos (2007), the tutor was configured to occasionally introduce intentional mismatches (e.g., slightly incorrect parameters, incomplete constraints, or plausible-but-not-fully-correct steps) so that students would have to test, notice inconsistencies, and debug rather than simply accept an output. This design choice was meant to make debugging episodes more likely and to keep mathematical control and validation with students. After an initial introduction by the authors on goals, workflow, and tools, each group received a booklet containing the task description, practical guidelines for working with the tutor (e.g., ask for small changes, specify constraints, request explanations of geometric choices), and a worksheet to document planning, intermediate attempts, and reflections. Students then began by discussing within the group and sketching an idea for a geometric logo. The task required two deliverables: a static design and a dynamic version obtained by introducing controlled changes over time. The work unfolded in two main phases. In Phase 1 (static logo), groups translated their sketch into a Turtle drawing by identifying the geometric elements involved (basic primitives, polygons,

repeated patterns, rotations/symmetries) and by making choices about parameters (step length, angles, number of repetitions, starting position). Students interacted with the customised ChatGPT tutor to generate initial code fragments, clarify Turtle commands, and adjust geometric constraints. They executed each attempt in a Python development environment that allowed them to run the script and immediately visualise the drawing, making it easy to compare intended and obtained outputs. In Phase 2 (animation), groups extended the static figure into a simple animation by introducing iteration and parameter variation (e.g., progressive rotations, incremental translations, repeated redraws, or stepwise changes to key parameters). This phase increased the need for explicit specifications because small changes in parameters could produce large visual effects. Groups typically worked through repeated cycles: formulate a prompt, obtain or modify a code fragment, run it, inspect the output, identify mismatches, and refine either the prompt (by adding constraints or clarifying the goal) or the code (by adjusting parameters and structure). Throughout both phases, students filled in the worksheet by recording design decisions, notable breakdowns, and the reasons behind revisions, providing a trace of their prompt–debug–refine process alongside the final digital artefact.

Data collection and analysis

Given the qualitative nature of the study, we adopted a multi-modal data collection strategy to capture both students' interaction with the GenAI tutor and the evolution of their artefacts. Data sources for each group included: i) digital interaction logs, namely the full prompt–response history with the customised ChatGPT tutor; ii) student worksheets, used to document initial planning, intermediate decisions, encountered breakdowns, and short reflections; and iii) digital artefacts, including the Python Turtle scripts, intermediate outputs/screenshots, and the final static and animated products. Analysis followed a qualitative interpretative approach and was conducted through triangulation across data sources: interaction logs were examined to map prompt–debug–refine cycles, with particular attention to how students noticed and handled mismatches (including those intentionally introduced by the tutor) and how their prompts evolved as specifications; worksheets were analysed to connect students' stated intentions and reflections with the observed interaction trajectories; the artefacts were inspected to corroborate the reconstructed process and link revisions to concrete changes in the produced geometry and animation. Episodes were then interpreted through the analytical categories defined in the theoretical framework to characterise how students' mathematical engagement was sustained during GenAI-mediated work.

RESULTS

The excerpts were selected to make visible how students' work evolved across prompt–run–inspect–revise cycles, focusing on: a) the prompts as written specifications of geometric intent, b) the group discussions triggered by mismatches between the intended and the rendered artifact, and c) the subsequent refinement of prompts after negotiating constraints (e.g., angles, repetition rules, positioning) within the group. In

line with the worksheet structure, we report short excerpts from the prompts (translated from Italian and lightly edited for readability) together with short reconstructed excerpts of in-group talk captured in field notes, and we connect them to the intermediate and final artifacts produced in the Turtle environment. According to the task design, groups first sketched an idea and articulated key constraints, then asked the GenAI tutor for Python Turtle code, executed it, and iterated whenever the output did not match the initial plan, documenting changes and motivations in the worksheet. For instance, Group1 aimed to design a logo based on a central pentagon combined with a fivefold symmetric star-like structure. Their initial request to the tutor was formulated in global visual terms and included only partial constraints: "*We want a logo with a pentagon in the center and five colored triangles around it, like a star. Add letters at the outer tips. Please give Python Turtle code*".

After running the code, the group compared the rendered image to the sketch and identified mismatches related to rotational spacing and component alignment. This triggered a local debugging discussion, where students moved from an aesthetic judgement ("it looks wrong") to explicit geometric constraints:

G1S1: The triangles overlap! This can't be five equal parts.

G1S2: If it's five, the turn must be 360/5. That's 72°.

G1S3: And the triangle should start from each side of the pentagon, not from a random point.

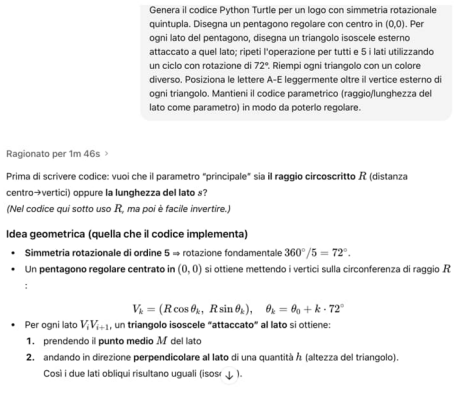
G1S2: Also, the letters must be at the triangle tips, not near the center.

Following this exchange, the group reformulated the request by turning the intended structure into checkable constraints (rotation angle, repetition, attachment rule to polygon sides, and label placement):

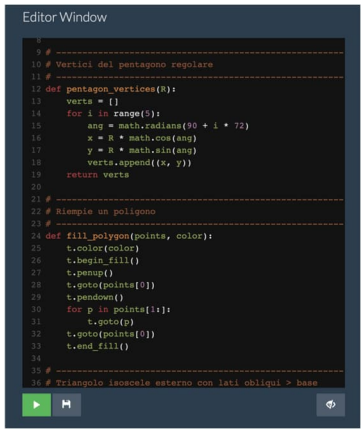
> *Please generate Python Turtle code for a logo with fivefold rotational symmetry. Draw a regular pentagon centered at (0,0). For each side of the pentagon, draw an external isosceles triangle attached to that side; repeat this for all 5 sides using a loop with rotation 72°. Fill each triangle with a different color. Place the letters A–E slightly beyond the outer vertex of each triangle. Keep the code parametric (radius/side length as a parameter) so we can adjust it.*

Figure 1: Star–pentagon logo design through GenAI and Turtle

a.



b.



c.

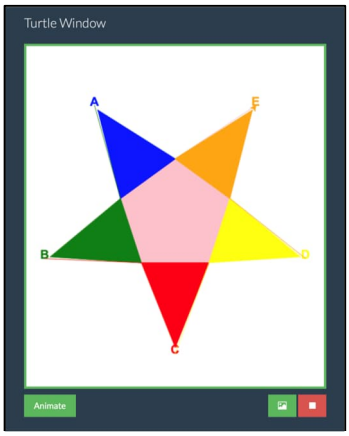


Fig. 1a reports the refined prompt sent to the customised tutor. Fig. 1b shows the corresponding implementation in the Python Sandbox editor, where the constraints are operationalised through modular functions (e.g., computing pentagon vertices from R and filling polygons) and a five-step loop. The resulting rendered artefact is shown in Fig. 1c, displaying the central pentagon, the five coloured external triangles, and the A–E labels positioned near the outer vertices.

A second case illustrates a different pathway: Group2 designed a representational logo (a wolf profile) and treated the figure as a sequence of directed segments. Here, the mismatch-driven refinement focused on decomposition into parts and on controlling the pen state (drawing vs repositioning). The initial request already contained an implicit decomposition ("outline + eye"), but lacked an operational description of how to connect parts without unwanted traces: "*We want to draw a wolf head profile with Turtle. Make a simple outline with straight segments, and add an eye. Use a scale parameter so we can adjust sizes.*"

After running the code, students noted two recurring issues: (i) some segments did not produce the expected "sharpness" and (ii) repositioning for the eye created unwanted lines. Their discussion turned these observations into procedural constraints:

G2S4: The ear is too round—our angles are not sharp enough.

G2S5: We should decide the headings: one segment, then turn, then another segment.

G2S4: Let's ask for a list of segments with setheading/forward, and penup/pendown for details.

The refined request translated the sketch into an executable "segment plan", explicitly asking the tutor to map geometric decisions into code primitives:

> *Write Python Turtle code to draw a wolf profile outline as a polyline. Use a scale parameter in cm and implement the outline as a sequence of steps using setheading(angle) + forward(length*cm). Keep the line color green and make the outline continuous. Then add a small filled circle for the eye: use penup() to move to the eye position without*

drawing, then pendown() and begin_fill()/end_fill() for the circle. Comment the code by naming key segments (snout, ear, neck).

Figure 2: Wolf logo students' design cycle

a. b. c.

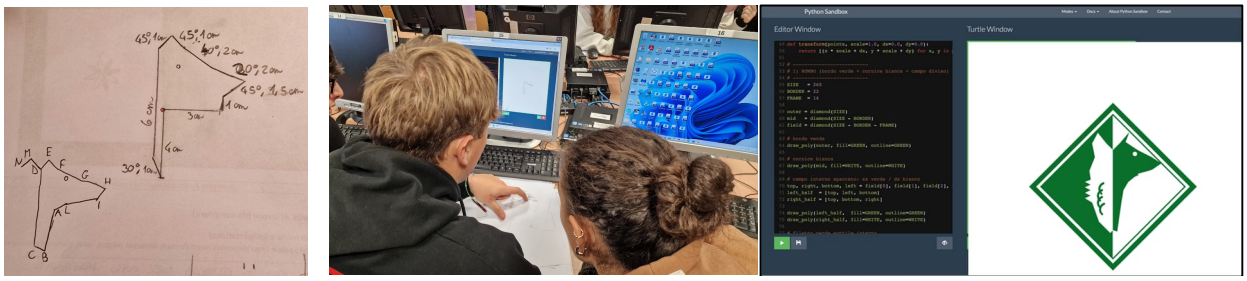


Fig. 2a documents the group's initial decomposition of the wolf profile into a sequence of straight segments, annotated in a sketch with estimated lengths and turning angles. Fig. 2b captures the group discussion in which students compared the sketch with the on-screen output and negotiated which parameters to modify (e.g., local angles for the ear and snout, segment lengths, and the need to separate drawing from repositioning). Fig. 2c shows the final implementation in the Python Sandbox environment: the editor snapshot makes visible how the agreed constraints were translated into code through explicit headings and scaled lengths, together with pen control to add non-outline details such as the eye without producing unwanted lines.

DISCUSSION AND CONCLUSIONS

The results address our research question by showing how learning by prompting can support students' mathematical thinking during a Turtle-geometry programming task. In both episodes, students' work developed through recurring prompt–run–inspect–revise cycles: groups compared the on-screen output with their intended design, identified a mismatch, and then negotiated which geometric constraints needed to be stated more clearly. In the star–pentagon logo, this meant making explicit fivefold symmetry (e.g., a 72° rotation step), attachment rules, and positioning constraints; in the wolf logo, it meant translating a sketch into a segment-by-segment plan with explicit headings, scaled lengths, and pen control for repositioning. Across these cases, the key shift was from asking for "a figure that looks like…" to producing prompts that functioned as mathematical specifications that could be tested and revised. This supports the paper's main claim: learning by prompting can be read as an extension of constructionist learning by making. The "making" remains visible in the executable code and the produced drawing, but prompting becomes part of the constructive work because students progressively transform an initial idea into constraints that are checkable on screen. In this sense, prompts do not replace making; they become a medium through which students articulate, refine, and test mathematical structure while programming. The findings also clarify what computational thinking looked like in this setting, without reducing it to generic "coding skills". In the star–pentagon case, students' revisions showed decomposition into components and parameter choices tied to symmetry and repetition; in the wolf case, students decomposed the outline into

controllable steps and used explicit commands to separate drawing from movement. Across both episodes, debugging primarily involved locating the mathematical source of a mismatch (angles, repetition rules, proportions, positioning) and revising the prompt accordingly. Furthermore, from a PBL perspective, the task sustained the prompt–debug–refine process because it combined (i) a shared and meaningful goal (designing a static logo and a simple animation), (ii) students' responsibility for choosing strategies and revising attempts, and (iii) collaboration as a condition for negotiating constraints and next steps. The immediate visual feedback of the Turtle environment made iteration inevitable and productive, while the worksheet served as a trace tool, capturing plans, intermediate attempts, and reasons for revisions: it required groups to externalise their initial plan, record successive changes, and justify revisions in relation to geometric properties (e.g., symmetry, angles, repetition rules, proportions, headings and pen control). In the end, this classroom-based qualitative case study suggests that, when a GenAI assistant is used to support iterative specification and checking, students' interaction can shift from accepting generated outputs to refining constraints and procedures. The contribution is not that GenAI "helps" in general, but that learning by prompting can keep mathematical reasoning central in programming activity by making constraints explicit, testable, and revisable.

Acknowledgment:

This work was developed in the context of the project STEAiM CT Empowering Pre-Service STEM Teachers with AI for Computational Thinking (2025-1-SK01-KA220-HED-000362022), partially funded by the Erasmus+ grant program of the EU.

References

Abelson, H., & diSessa, A. A. (1986). Turtle geometry: *The computer as a medium for exploring mathematics*. MIT Press.

Bernardi, M. L., Capone, R., & Lepore, M. (2025). Empowering mathematics educators: Integrating ChatGPT as a tool for innovative teaching practices. In B. du Boulay, T. Di Mascio, E. Tovar, & C. Meinel (Eds.), *Proceedings of the 17th International Conference on Computer Supported Education*, (pp. 404–411).

Hmelo-Silver, C. E., & Barrows, H. S. (2006). Goals and strategies of a problem-based learning facilitator. *Interdisciplinary Journal of Problem-Based Learning, 1*(1), 21–39.

Kynigos, C. (2007). Half-baked microworlds as boundary objects in integrated design. International *Journal of Computers for Mathematical Learning, 12*(2), 87–111.

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. Basic Books.

Pepin, B., Buchholtz, N., & Salinas-Hernández, U. (2025). A scoping survey of ChatGPT in mathematics education. *Digital Experiences in Mathematics Education, 11*(1), 9–41.

Walkington, C. (2025). The implications of generative artificial intelligence for mathematics education. School Science and Mathematics. Advance online publication.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35.