Journal of Technology-Integrated Lessons and Teaching
(2026) Volume 5, Issue 1; DOI: 10.13001/jtilt.v5i1.10307

Building AI Capacity Through Coding Fundamentals: AI-Augmented Coding for Teens Using Python, Data Structures, and Verification Practices

Sidney Shapiro, University of Lethbridge

Overview

This activity helps secondary learners (grades 7–12) build artificial intelligence (AI) capacity by using a chatbot to support, not replace, foundational Python programming. In a 75–90 minute coding activity, learners choose a short project that uses lists and/or dictionaries, write a specification and test plan, ask the chatbot for a plan before coding, implement and debug the program, and verify each AI suggestion through tests, tracing, and explanation. The purpose is to connect tool awareness, prompt writing, data structures, and responsible use in one reusable activity. Major assessments include a runnable program, a Prompt and Verification Log, and a brief explanation/reflection. Setup requires 10–15 minutes plus any local approval for youth chatbot access.

Topics: AI Literacy, Python Programming, Computer Science, Secondary Education

Time: One 75-90-minute session.

Materials

- Learner devices with a Python environment, such as local Python, a browser-based IDE, or Jupyter.
- Projected display for instructor modeling.
- Access to an LLM chatbot. If individual accounts are restricted, use an instructor-controlled account, a school-managed tool, or the offline AI-output-card described in the facilitation guide.
- Prompt and Verification Log.
- Instructor Facilitation Guide.
- Assessment Rubric.
- Project Prompts and Code Skeletons.

Context-at-a-Glance

Setting
A multi-day camp for secondary learners (grades 7-12) in southwestern Canada.

Modality
Face-to-face; adaptable online with screen sharing, structured check-ins, and shared logs.

Class Structure
75 to 90-minute insertable activity drawn from multi-day camp blocks; pairs or triads rotate production, documentation, and verification roles.

Organizational Norms
No sensitive data entered into third-party tools; AI use documented throughout the learning process.

Learner Characteristics
Grades 7-12 with mixed coding experience; all 44 learners completed the 2024 and 2025 programs.

Instructor Characteristics
Instructor should know introductory Python, debugging, and coaching through questions.

Development Rationale
Fast AI-supported prototypes, especially in 2025, made explicit data-structure reasoning, tests, tracing, and explanation necessary before accepting AI output.

Design Framework
AI-adapted PRIMM, UDL-aligned options, collaboration roles, and responsible AI literacy practices.

Setup

Before class, the instructor verifies that Python runs on all devices, confirms local policy for youth access to AI tools, and prepares one short demonstration showing a chatbot response that looks plausible but fails a test. The instructor prints or posts the prompt and post verification log, instructor facilitation guide, assessment rubric, and project prompts and code skeletons and decides whether starter code will be available at the beginning of the session or released after learners have attempted the task.

Setup typically takes 10–15 minutes after account access and classroom technology are available. If chatbot access is unavailable or inappropriate for the setting, the same activity can be implemented with printed "AI output cards" containing sample plans, code, and intentional defects.

Standards Alignment

The activity can be aligned to local computer science, digital literacy, and technology integration outcomes. For broad transfer, it also aligns with UNESCO's AI Competency Framework for Students through

- Attention to human agency,
- Responsible tool use,
- Critical evaluation of AI outputs,
- Practical AI literacy (UNESCO, 2024b).

Context and Setting

Generative AI has made natural-language-to-code workflows accessible to beginners, but novices can struggle to write useful prompts, interpret generated code, and judge whether a response is correct (Nguyen et al., 2024). Recent computing education work similarly emphasizes that LLM use should shift attention toward decomposition, explanation, testing, and debugging rather than only syntax production (Vadaparty et al., 2024; Yeh et al., 2025). For teen learners, this means that AI capacity should not be defined as the ability to obtain code quickly. Rather, AI capacity should be defined as the ability to work with AI tools while understanding the data structures, logic, tests, and design choices that make code usable.

This activity emerged from coding camps for grades 7–12. The 2024 implementation, delivered in partnership with a public library and a Canadian public university, included 19 participants. The 2025 implementation, delivered at the university, included 25 participants. The AI-specific routine was used in both years, although learners' use of AI coding support became more visible and productive in 2025 as tools became more available.

Across the camps, learners built stock market simulators, choose-your-own-adventure games, games using dictionaries and functions, projects using external Python libraries, and map-based games that read room data from JSON or CSV files. This activity distills the reusable instructional routine that emerged from those implementations.

The activity is organized around an AI-adapted PRIMM sequence. Learners:

1. Predict what code (or an AI suggestion) should do,
2. Run it,
3. Investigate mismatches,
4. Modify the code with bounded AI assistance,
5. Make a small working feature (Sentance et al., 2019).

Instructor modeling and coaching are important during these tasks, but the activity is not presented as a full cognitive apprenticeship model. Instead, it uses modeling, role rotation, and brief articulation/reflection moments to make the normally hidden parts of AI-assisted coding visible: specification, data representation, tests, tracing, and decisions to use, modify, or reject an AI suggestion.

The activity also uses Universal Design for Learning (UDL) to support learner variability. Learners may choose among project prompts; use starter code when syntax load blocks reasoning; and demonstrate learning through code, a flowchart, or a brief spoken explanation (CAST, 2024). AI literacy is operationalized through observable actions: Identifying appropriate and inappropriate uses, writing bounded prompts, evaluating outputs, documenting assistance, and making a use/modify/reject decision before integration (Lee et al., 2024; Ng et al., 2021). Responsible AI practices are embedded through privacy rules, disclosure, and verification expectations (Miao & Holmes, 2023; National Institute of Standards and Technology, 2023; OECD, 2026).

Learning Representation

Learning objectives

- Identify at least two appropriate and two inappropriate uses of a chatbot as a coding support tool.
- Choose and justify a list and/or dictionary representation for a small Python task.
- Write a bounded prompt that requests a plan, assumptions, and test cases before requesting code.
- Analyze an AI suggestion for likely problems, then accept, revise, or reject it using evidence from tests, tracing, or explanation.
- Explain one design decision and one debugging or verification step to a peer or instructor.

Activity design and flow

Learners work on the activity (see Table 1) in pairs or triads using simplified collaboration roles:

1. The Driver types and runs the code.
2. The Navigator reads the specification and maintains the Prompt & Verification Log.
3. When a third learner is present, the Verifier predicts output, checks tests, and challenges unsupported AI suggestions.

Roles rotate every 10–15 minutes so that every learner practices both production and reasoning.

Time	Phase	Instructor moves	Learner actions and artifacts
0–10 min	Orientation: tool role and norms	Demonstrate a chatbot response that looks correct but fails a test. Review privacy, attribution, and "verify before you trust." Refer learners to the Prompt and Verification Log.	Review the Prompt and Verification Log and choose Driver/Navigator/Verifier roles.
10–20 min	Problem framing before AI	Model how to extract requirements, choose a list/dictionary representation, and write 3–5 tests including one edge case.	Select a project prompt, draft a specification, sketch a data model, and write initial tests.
20–30 min	Prompt for a plan, not code	Provide the plan-prompt template from the Instructor Facilitation Guide and require assumptions and test cases before code.	Prompt for a plan, compare AI suggestions with their own plan, and record decisions in the log.
30–55 min	Implementation and guided debugging	Circulate and coach with questions: "What key does the dictionary use?", "Which test failed?", "What line changes the list?" Release starter code only when syntax blocks the data-structure objective.	Implement one function or feature at a time. Ask the chatbot only for bounded help, such as one bug or one function.
55–70 min	Verification checkpoint	Stop the room for a short process check. Model a trace print or assertion. Ask pairs to identify one AI suggestion they accepted, revised, or rejected.	Run tests, add one edge case, trace one loop or dictionary update, and complete the verification rows in the Prompt and Verification Log.
70–90 min	Articulation and reflection	Facilitate 2–3 process shares focused on evidence, not polish. Use the Assessment Rubric criteria.	Submit the runnable program and log; explain one design decision and one verification step in writing or verbally.

Table 1. Activity flow and timing.

The instructor should describe these roles as a way to distribute attention, not as a rigid professional pair-programming model.

The instructor may use one of three project prompts from the Project Prompts and Code Skeletons: A camp inventory manager, a text-adventure micro-scene, or a quiz data model for a later web app. Each prompt requires learners to use lists and/or dictionaries. The instructor models one prompt, then either assigns that prompt to all groups for consistency or allows choice when learners are ready for more autonomy.

What Verification Looks Like

Verification is the central learning move. After a chatbot suggestion, learners pause before copying the change into the final program:

1. They predict what the code should do using a concrete input.
2. They run at least three tests, including one edge case such as an empty list, missing dictionary key, invalid menu option, or quantity that would go below zero.
3. They trace the relevant state change, for example by printing the dictionary before and after an update.
4. They decide whether to use, modify, or reject the AI suggestion.

For example, if a chatbot suggests using len(inventory) to count total camp supplies, learners test the starter dictionary {"water": 10, "snack": 5}, notice that the expected total is 15 rather than 2, and modify the function to sum the dictionary values. The error becomes evidence for learning rather than a failure to hide.

Starter Code and Scaffolding Decisions

Starter code is a scaffold, not the default path for every learner. It may be provided at the beginning of the activity for groups with very little prior coding experience or released after 5–7 minutes if a group can explain the data model but is blocked by syntax. It should be withheld or partially hidden for groups ready to design independently. When starter code is used, the instructor should still require learners to fill in the specification, tests, and verification log. This preserves the learning objective: Students may receive structure, but they still need to reason about the data and verify the code.

Differentiation and Accessibility

The activity includes multiple entry points for mixed-readiness teen groups. Project choice supports motivation, while a common rubric maintains shared expectations. Learners who need reduced syntax load may use starter code or prompt stems; advanced learners may add input validation, file persistence, summary statistics, or a small Flask connection. Learners may submit a brief oral explanation, annotated code comments, or a simple flowchart in place of a longer written reflection. A no-AI path is also possible: The instructor could distribute sample AI responses or defective code cards for learners to verify and repair. This option supports settings where youth access to chatbot accounts is restricted.

Evaluation

Evaluation focuses on observable evidence rather than tool novelty. The core indicators of success are:

- The program uses an appropriate list and/or dictionary;
- The learner group provides at least three tests and one edge case;
- The Prompt and Verification Log documents at least one AI-assisted suggestion and a use/modify/reject decision;
- Learners can explain one design decision and one verification step.

The Assessment Rubric provides additional details. In the completed camp implementations, all participants finished the program (19 in 2024 and 25 in 2025), and informal student feedback was positive. Formal pre/post assessment data were not collected; therefore, the activity should be interpreted as an implemented teaching activity with formative indicators rather than an efficacy study. Future implementations should collect brief pre/post AI-literacy reflections and completion indicators to strengthen evaluation evidence.

Critical Reflection

This activity was refined from two completed teen coding camps and from the development of expanded 2026 camp resources. In 2024, learners were highly engaged by themed projects and built ambitious prototypes, including simulations and branching games, but progress was slower and instructors needed to provide more direct syntax and debugging support. In 2025, more accessible AI coding tools allowed learners to build more quickly, which expanded what was possible in a short camp but also made it easier for learners to accept code they could not yet explain. This contrast shaped the activity's current emphasis: AI can increase building capacity, but foundational concepts such as lists, dictionaries, functions, file structures, tests, and tracing remain necessary if learners are to work with AI rather than be replaced by it.

Instructors observed that the log and checkpoint helped shift some learner conversations from authority-based reasoning toward evidence-based reasoning. Without the log, some learners treated chatbot output as final. With the log and checkpoint, conversations more often focused on whether tests passed, which dictionary key changed, or why a list did not contain the expected item. This implementation observation supports making verification a required artifact rather than a reminder. It also supports keeping the activity small: One function, one data model, and one verified feature are enough for a meaningful 75-90 minute activity.

Two practical challenges should be anticipated. First, camp settings include varied attention, uneven prior knowledge, and time pressure. The activity works best when the instructor keeps scope small and treats extra features as extensions. Second, AI access for minors can be inconsistent. The offline output-card option is therefore not a backup of lesser value; it is a useful way to teach the same verification habits without requiring individual chatbot accounts. The next planned modification is a train-the-trainer version for peer mentors and camp assistants, plus an open educational resource package containing editable prompt cards, starter projects, and facilitator notes.

References

CAST. (2024). Universal Design for Learning Guidelines 3.0. https://udlguidelines.cast.org/

Lee, K.-w., Mills, K., Ruiz, P., Coenraad, M., Fusco, J., Roschelle, J., & Weisgrau, J. (2024, June 18). AI literacy: A framework to understand, evaluate, and use emerging technology. Digital Promise. https://digitalpromise.org/2024/06/18/ai-literacy-a-framework-to-understand-evaluate-and-use-emerging-technology/

Miao, F., & Holmes, W. (2023). Guidance for generative AI in education and research. UNESCO. https://unesdoc.unesco.org/ark:/48223/pf0000386693

National Institute of Standards and Technology. (2023). Artificial Intelligence Risk Management Framework (AI RMF 1.0) (NIST AI 100-1). https://doi.org/10.6028/NIST.AI.100-1

Ng, D. T. K., Leung, J. K. L., Chu, S. K. W., & Qiao, M. S. (2021). Conceptualizing AI literacy: An exploratory review. *Computers and Education: Artificial Intelligence, 2,* Article 100041. https://doi.org/10.1016/j.caeai.2021.100041

Nguyen, S., Babe, H. M., Zi, Y., Guha, A., Anderson, C. J., & Feldman, M. Q. (2024). How beginning programmers and Code LLMs (mis)read each other. In Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24). ACM. https://doi.org/10.1145/3613904.3642706

Organisation for Economic Co-operation and Development. (2026). *OECD digital education outlook 2026: Exploring effective uses of generative AI in education*. OECD Publishing. https://doi.org/10.1787/062a7394-en

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

U.S. Department of Education, Office of Educational Technology. (2023). *Artificial intelligence and the future of teaching and learning: Insights and recommendations*. https://tech.ed.gov/ai-future-of-teaching-and-learning/

UNESCO. (2024a). AI competency framework for teachers. https://doi.org/10.54675/ZJTE2084

UNESCO. (2024b). AI competency framework for students. https://doi.org/10.54675/JKJB9835

Vadaparty, A., Zingaro, D., Smith IV, D. H., Padala, M., Alvarado, C., Benario, J. G., & Porter, L. (2024). CS1-LLM: Integrating LLMs into CS1 instruction. In Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024). ACM. https://doi.org/10.1145/3649217.3653584

Yeh, T. Y., Tran, K., Gao, G., Yu, T., Fong, W. O., & Chen, T.-Y. (2025). Bridging novice programmers and LLMs with interactivity. In Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS 2025) (pp. 1295–1301). ACM. https://doi.org/10.1145/3641554.3701867

About the Author

Sidney Shapiro is a professor of business analytics in the Dhillon School of Business at the University of Lethbridge. His work focuses on applied data science, social network analytics, and the practical integration of artificial intelligence into learning and organizational settings. He designs technology-rich learning experiences that combine computational thinking, tool literacy, and responsible use of AI. He leads teen coding camp programming associated with the University of Lethbridge, with materials developed across 2024–2026 that support pathways from introductory Python to web applications, software development, and emerging technology topics. He can be contacted regarding this article at sidney.shapiro@uleth.ca

Sharing & Modification Permissions