# Reflecting on Knowledge and Learning: Revisiting Plato's Meno Through Game Programming with Python

Marcus Birkenkrahe

November 12, 2024

Reflecting on teaching game programming in Python, Plato's *Meno* brings up essential questions about knowledge: "How do we know what we know?" and "How do we recognize the need for further learning?" Like Meno questioning Socrates, I find myself revisiting my own approach to learning and teaching. As Donald Knuth puts it, "Everyday life is like programming. If you love something, you can put beauty into it" (Knuth, 1997). This perspective—an approach that values clarity and depth—has been central to my teaching, shaped by years of coding and my commitment to literate programming, which sees code not just as technical instructions but as a thoughtful expression of ideas.

## Game Programming with Python: Beyond Syntax and Structure

Teaching game programming with Python is more than syntax; it's about sharing a cognitive framework for problem-solving and creativity. Python offers a medium for exploring logical structures, and my students' early projects—such as a number-guessing game—introduced them to the fundamentals of programming: variables, loops, and conditional statements. These projects are more than technical exercises; they foster an evolving way of thinking, much like the self-reflective approach Socrates promotes in his dialogues (Plato, trans. 1994).

My own programming journey began with languages like FORTRAN and BASIC, a background that influences how I teach today. I vividly remember being captivated as a teenager by a 3D maze game on the TI-99, struck by the idea that code could bring concepts to life. Today, I see that same spark in my students, who approach Python as a language that enables them to create something meaningful. I emphasize literate programming techniques (Knuth, 1984) to help students see programming as an extension of their thought processes. For them, documenting logic and expressing intent within their code makes their work both functional and readable.

## Struggles and Milestones: Navigating the Growing Pains of Learning and Teaching

Learning to program, like learning any skill, has its frustrations. Teaching it reminds me of the "learning paradox" Socrates raises in /Meno/—the tension between realizing something isn't fully understood and the struggle to make sense of it (Plato, trans. 1994). Debugging sessions can be frustrating, yet they model perseverance and attention to detail, qualities fundamental to problem-solving. Debugging mirrors the "back and forth" questioning in the Socratic method, where each error and correction brings the learner closer to clarity (Vlastos, 1991).

These learning struggles often bring a mix of frustration and reward. Teaching students how to debug a syntax error or troubleshoot conditional logic reminds me of my own early days with FORTRAN, where one small typo could disrupt an entire calculation. I find these experiences reinforce Knuth's view that love for one's work brings beauty, even to the meticulous process of debugging (Knuth, 1997). Through each student's process, I am reminded that programming offers as much insight as it does skill, merging functionality with an elegance grounded in logic.

## Markers of New Learning: Recognizing Growth

How do I recognize that I've learned something new? Often, it begins with clarity after confusion, when a previously opaque concept becomes intuitive. For example, when students move from struggling with basic loops to implementing effective game logic, they reveal their deepened understanding. The value of literate programming (Knuth, 1984) is to document this learning journey, helping students express their ideas clearly in code and in words, as each "Aha!" moment is a marker of genuine growth.

As students progress, I see their questions shift from syntax to design and purpose, a shift I encourage through literate programming techniques. By documenting their logic, they see how their thinking evolves. For me, this aligns with Dewey's concept of experiential learning, which stresses learning through reflection and iterative improvement (Dewey, 1938). Every time a student grapples with a new concept or revisits an idea, it reinforces that knowledge is not static but continuously built upon.

## Looking Forward: What Lies Ahead in Game Programming and Pedagogy

Programming—and teaching it—will always involve ongoing learning. Topics like object-oriented design and algorithmic efficiency are the next steps in my course, concepts that open doors to more complex games and refined structures. And as I introduce students to these concepts, I continue to draw on literate programming to make code not just functional, but clear and expressive (Knuth, 1984).

In his reflection on knowledge, Socrates suggests that understanding involves a willingness to continually seek answers and to reframe one's assumptions. This approach, coupled with Knuth's appreciation for the beauty in one's work, guides me. Teaching is an iterative journey that demands openness, humility, and appreciation for the process of discovery, especially in programming, where knowledge constantly evolves (Knuth, 1997; Plato, trans. 1994).

Ultimately, I approach teaching and learning as a lifelong dialogue, one that requires both curiosity and commitment. Like Socrates, I find that teaching is about asking better questions, and, like Knuth, I believe that beauty can be found in the work itself. In programming, as in life, learning is about refining our inquiries, cultivating understanding, and embracing both the struggle and the elegance of discovery.

## References

- Dewey, J. (1938). *Experience and Education.* Collier Books.
- Knuth, D. E. (1984). *Literate Programming.* Center for the Study of Language and Information.
- Knuth, D. E. (1997). *Selected Papers on Computer Science.* Center for the Study of Language and Information.
- Plato. (1994). *Meno* (G.M.A. Grube, Trans.). In *Complete Works* (J.M. Cooper, Ed.). Hackett Publishing.

- Vlastos, G. (1991). *Socrates: Ironist and Moral Philosopher.* Cornell University Press.