Wolfram Language for Teaching Computational Thinking to K-12 Learners

Alyson Gamble, Wolfram Research, Champaign, IL, USA

ABSTRACT

Computational thinking is a necessary skill, but developing it among young learners can be a difficult process. Wolfram Research, known in part for its accessible Wolfram Language and complex computational tool Mathematica, has recently developed new initiatives for introducing computational thinking to novice learners. These tools help make computational thinking accessible to students across the K12 curriculum.

KEYWORDS

Computational Thinking, K-12 Education, Novice Learners, Open Source Software, Programming

INTRODUCTION

Introducing digital technology into the school curriculum has drawn several competing names for what is essentially a new underlying discipline at K-12 level, including computer science, informatics, and computational thinking. According to Janette Wing (2006), the term computational thinking “…involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.” She labored the point that computational thinking was a “universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.”

Computational thinking is now considered an important part of learning development. Yet it can be difficult for novice learners to gain computational thinking skills. In order to introduce computational thinking in an accessible manner, instructors and mentors must pay attention to students’ interests while using educational tools that have a low threshold and high ceiling. The Wolfram Language, a knowledge-based programming language, fulfills this need for an accessible yet sophisticated language to introduce novice learners to computational thinking. Newly developed tools at Wolfram Research can help provide introductions for learners to the Wolfram Language and engage them in computational thinking.

BACKGROUND

In 1962, envisioning how machine automation would impact the world, Alan Perlis asserted that programming should be part of a liberal education (Guzdial, 2008, p. 25). Steve Jobs, who many
considered a visionary in the digital world, said as much when asked in an interview what he did or used programming for when he was an adolescent: “(You use) it to be a mirror of your thought processes. To learn how to think. I think everybody in this country should learn how to program a computer. To learn how to think” (Jobs, 1995).

Programming, or coding as some in curriculum design now prefer to call it, is one way to develop and utilize computational thinking, but programming is not the only purpose of computational thinking, and should arguably be introduced before learners understand basic computational thought processes (Fletcher & Lu, 2009, p. 24; Grover & Pea, 2013, p. 40). Computational thinking, or problem solving by framing questions in a way that can be communicated to a computer, is essentially “thinking like a computer scientist when confronted with a problem” (Grover & Pea, 2013, p. 39). A computational thinker can efficiently and systematically process information and tasks (Fletcher & Lu, 2009, p. 23). Computational thinking is not just applicable to computer science; it allows for innovation in other fields, including business and the fine arts (Grover & Pea, 2013, p. 39).

Bringing computational thinking to K-12 education was originally suggested in the early 1980s by Seymour Papert, leading to a significant pedagogical foundation, but it has not become a required piece of many country’s curriculum (Grover & Pea, 2013, p. 38). With packed schedules, K-12 educators often lack the time to integrate what some consider a completely new concept. Additionally, there is some misunderstanding about computational thinking itself and how it differs from other science, technology, engineering, and math (STEM) thinking and frameworks. Computational thinking is distinct from other STEM frameworks because it relies on information processes. In part because computational thinking has been associated with academic computer science departments rather than K-12 classrooms, the majority of research done on computational thinking has been conducted in undergraduate environments (Grover & Pea, 2013). Nonetheless, it has considerable potential to loosely integrate all of those subjects and more, as a foundational subject in a forward-thinking curriculum.

Meanwhile, research into computational thinking in the K-12 environment does contain recommendations for how to introduce its concepts to new learners. Especially when working in K-12, instructors and mentors can easily overstep and break a child’s interest in learning code. Therefore, it is important for the children to lead the initiative by directing their learning and focusing on issues that are interesting to them (Good, 2011, p. 18; Goschnick, 2015, p. 50). In higher education, this means finding the bridge between computational thinking and disciplinary issues (Guzdial, 2008, p. 27). For younger learners, the focus turns to less formal, more personal interest, such as storytelling (Good, 2011, p. 18) or class assignments (Wolfram, 2016).

Introductory computational thinking tools need have a low threshold and a high ceiling (Grover & Pea, 2013, p. 40). The low threshold means that when these young users are able to quickly see results of their activities, then they are motivated to continue engaging with the programming environment (Good, 2011, p. 18). Once students have mastered the basics, they need a high ceiling, meaning the learner must be able to create sophisticated material, such as a complex game (Reppening, Webb, & Ioannidou, 2010, p. 266).

Curricular tools for computational thinking must follow a clear process and lessons learned from them should be able to transfer to other environments. A tool’s flow must scaffold, where each step has associated skills and challenges. Each step is well-designed and, once mastered, leads the learner to a more complex computational thinking skill. Once the skills have been acquired in one area, learners should be able to transfer them to others. For example, if a student creates a game, then the computational thinking skills acquired during this experience must be applicable to work in a computer science class (Reppening, et al., 2010, pp. 267-268).