Purpose-first programming: Scaffolding programming learning for novices who care most about code's purpose

Kathryn Cunningham
School of Information
University of Michigan
kicunn@umich.edu

Abstract

Becoming "a programmer" is associated with gaining a deep understanding of programming language semantics. However, as more people learn to program for more reasons than creating software, their learning needs differ. In particular, end-user programmers and conversational programmers often care about code's purpose, but don't wish to engage with the low-level details of precisely how code executes. I propose the creation of scaffolding that allows these learners to interact with code in an authentic way, highlighting code's purpose while providing support that avoids the need for low-level knowledge. This scaffolding builds on theories of programming plans.

1. Semantics-focused activities don't meet the needs of certain learners

Introductory programming activities often echo the epigram *"To understand a program you must become both the machine and the program"* (Perlis, 1982). Commonly-used code tracing exercises (Sorva, 2013) ask students to simulate the execution of a program: determining the order in which lines of code run, which values are created and modified, and when the program starts and stops. Computing education researchers have created theoretical hierarchies of programming skills that describe code tracing as a primary skill that students should be taught before they learn to write code or explain the purpose of code (Xie et al., 2019).

However, for many programming learners, writing code or being able to read code and determine its purpose are far more important than understanding code semantics. In thinkaloud interviews about code tracing activities, I found learners who wanted to use code to solve problems but didn't view themselves as "a programmer" or didn't feel they can think "like a computer". These learners rejected code tracing activities, describing them as in conflict with their self-beliefs (Cunningham, Agrawal Bejarano, Guzdial, & Ericson, 2020). At the same time, these learners expressed a value for learning about code that solved "real" tasks, not toy problems.

2. An approach that emphasizes code's purpose

To meet the needs of these learners, I propose **purpose-first programming**, an approach to learning programming skills that starts from the user's need for programming, rather than the demands of programming language syntax and semantics. Instruction will focus on common programming patterns in the domain of choice, called plans (Soloway & Ehrlich, 1984). Purpose-first programming learners assemble and tailor plans, so that they can write useful programs in an area of interest right away, without needing to understand every detail of how code works.

2.1. Purpose-first programming scaffolds make programming easier

2.1.1. Code is grouped by plan

Plans are highlighted to allow learners to "chunk" (Gobet et al., 2001) programs more easily, hopefully leading to faster problem-solving. In purpose-first programming, plans are drawn from a specific domain. As a result, information about the plans can be described in domain-specific terms that facilitate the application of the plan to solve problems.

2.1.2. Plans consist of fixed code and "slots" that can contain objects or code

"Slots" are the only areas of a plan that can be changed. This decreases the complexity of editing code and draws learners' attention to the most important parts of code.

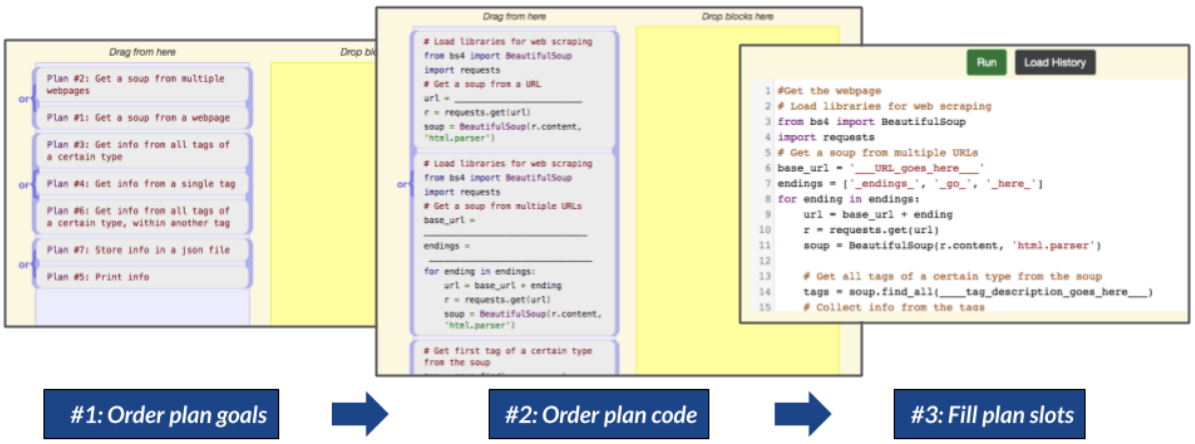

Figure 1 – Purpose-first code writing supports assembly and tailoring of plans.

2.1.3. Subgoals break plans into pieces to guide plan integration and tracing

Subgoals (Morrison, Margulieux, & Guzdial, 2015) provide a smaller unit of cohesive code, and a subgoal label clarifies the contribution to the plan's purpose. By tracing the input and output to each subgoal, learners can trace purpose-first code at a higher level of abstraction.

2.2. Designing a purpose-first programming proof-of-concept

I will design a proof-of-concept curriculum that teaches five plans from the domain of web scraping with BeautifulSoup. Learners will view examples of complete programs, learn about each plan individually, and then write, debug, and describe code that uses combinations of plans not previously seen. Activities will provide purpose-first scaffolds (see support for code writing in Figure 1).

2.3. How do learners describe the effect of purpose-first programming on their motivation?

2.3.1. Evaluating expectancy of success and value for tasks

I will perform semi-structured interviews with 6-12 novice programmers after they complete the purpose-first programming curriculum. According to the Eccles Expectancy-Value Model of Achievement Choice, motivation for a task is explained by expectancy of success on the task and value for the task (Eccles, 1983). The interviews will explore learners' feelings of success during activities in the curriculum, as well as expectations of success with similar learning in the future. The interviews will also ask learners about aspects of their value for this type of learning activity, including their enjoyment, alignment with future goals, and alignment with self-identity.

3. References

Cunningham, K., Agrawal Bejarano, R., Guzdial, M., & Ericson, B. (2020). I'm not a computer: How identity informs value and expectancy during a programming activity. In *Proceedings of the 2020 international conference of the learning sciences.* International Society of the Learning Sciences.

Eccles, J. (1983). Expectancies, values and academic behaviors. *Achievement and achievement motives*.

Gobet, F., Lane, P. C., Croker, S., Cheng, P. C., Jones, G., Oliver, I., & Pine, J. M. (2001). Chunking mechanisms in human learning. *Trends in cognitive sciences*, *5*(6), 236–243.

Morrison, B. B., Margulieux, L. E., & Guzdial, M. (2015). Subgoals, context, and worked examples in learning computing problem solving. In *Proceedings of the eleventh annual international conference on international computing education research* (pp. 21–29). ACM.

Soloway, E., & Ehrlich, K. (1984, September). Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, *SE-10*(5), 595–609.

Sorva, J. (2013, July). Notional machines and introductory programming education. *Trans. Comput. Educ.*, *13*(2), 8:1–8:31.

Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H., … Ko, A. J. (2019). A theory of instruction for introductory programming skills. *Computer Science Education*, *29*(2-3), 205–253.