THE KEY TO HELPING NOVICE PROGRAMMERS: **LANGUAGE ACQUISITION INSTRUCTION**

Introductory programming instruction can reach more, if not all, students if supplemented with pedagogies that address the acquisition of programming languages as languages per se

STORY BY Scott R Portnoff

Instructors of any introductory computer programming course routinely observe two groups of students: those who learn and progress (the traditional demographic overwhelmingly composed of white/Asian males) and a sizeable group who, though at grade level, struggle throughout and learn practically nothing. Known as the Novice Programmer Failure Problem (NPFP), over four decades of efforts to improve outcomes for the strugglers have proven unsuccessful. At the secondary level, the phenomenon is even more lopsided and intractable, affecting the majority of grade-level students. In the United States, secondary CS instructors (90% of whom it is estimated have no formal CS education), like all public school teachers, are tasked with delivering effective instruction to all students, thus putting them in an existential bind. About a dozen years ago, a small group of educators decided that programming education – i.e. the Advanced Placement Computer Science A course (APCS-A) – was simply not feasible.

The survey courses

As response, two secondary survey courses, Exploring Computer Science and Advanced Placement Computer Science Principles, were developed with National Science Foundation (NSF) funding. Proponents viewed them not as a solution to the NPFP, but as a way to skirt it altogether. Attacking the 'programming-centric' focus of the APCS-A course as exclusionary, they concocted a novel narrative in which programming was simply one of several co-equal strands in the secondary CS curriculum. Despite the rhetoric, both courses are considered pre-APCS-A courses and have proliferated in low-performing urban public high schools.

Proponents argued that the survey courses would 'broaden participation' beyond the traditional APCS-A demographic to females and traditionally under-represented minorities. NSF's goal was more pragmatic: to create a high school entry point into a CS pipeline intended to alleviate a massive number of projected computing vacancies.

At the same time, however, a College Board study found that students taking the APCS-A exam had a six- to eight-fold higher probability of choosing a CS college major (Morgan & Klaric, 2007). A later study found that 20% of students who score 2 or higher on the exam choose a CS major, and that fully 27% of students earning a 5 go on to major in CS (Mattern, Shaw, & Ewing, 2011). Thus the APCS-A course was identified as being the entry point into NSF's CS pipeline, an inconvenient fact ignored by the new narrative.

Redefining 'participation'

The survey courses intentionally avoided attempts to deliver rigorous programming

instruction – doing nothing to prepare students for subsequent CS coursework that would inevitably involve programming – opting instead for superficial hobby-like treatments of programming and other topics. As these courses proliferated, the effect over the past decade has been to greatly lower academic expectations for secondary CS education. Instead of helping students master content that vertically aligns with subsequent academic coursework, mere exposure to simplified topics is expected to generate interest that will magically enable students to later circumvent the NPFP and acquire programming competence.

Unlike APCS-A, there exists no research demonstrating that the survey courses spur students to take – or prepare them to pass – the APCS-A exam or future CS college coursework. Moreover, the survey course narrative that demoted programming from its central place in the introductory curriculum is a fiction. The universal post-secondary consensus is that programming is the core skill fundamental to the entire discipline – crucial for understanding and plumbing both basic and advanced CS topics including algorithms – on anything beyond a trivial level.

Instead of authentically 'broadening participation', therefore, the survey courses have simply swapped one unequal two-tier track system for another, ironically, but predictably, recapitulating the very inequities they were intended to remedy. The facade of more high school students taking classes labelled 'Computer Science' may have PR value, but it's only the latest in a long history of ineffective 'innovations' that continue to obstruct, confuse and delay real reform.

That being said, APCS-A is not without its own problems. The course may be the CS pipeline entry point, but it's simply too advanced for most grade-level students and demands a prerequisite. Why CS educators then put their efforts into developing a survey course is a mystery, because it's unremarkably obvious that an introductory course should be a programming course that will effectively identify and address head-on the problems novice programmers encounter.

MEMORISATION: AN IMPLICIT STRATEGY

Instructors routinely observe students struggle with syntax errors that block program compilation and hamper learning. Memorisation of small programs seemingly overnight facilitates the acquisition of the basic syntactic features necessary to avoid compiler errors. How? The repetition required for perfect memorisation bombards a learner's brain with idealised language data and patterns, priming it to inductively construct the programming language's syntax, as it does for natural languages.

Programming Language Acquisition
Automaticity
Practice
Repetitive Exposure
Meaningful Contexts
Intentional Expression
Feedback

The critical pedagogic role of language

In this regard, three recent fMRI studies have confirmed that comprehension of computer programs occurs in the same regions of the brain that process natural languages – not math, not logic (Siegmund et al, 2014) (Floyd et al, 2017) (Siegmund et al, 2017). This cognitive-physiological evidence indicates that programming languages, despite being artificial languages, are alive in the brains of programmers in much the same way as any natural language that those programmers speak. This is a profound paradigm shift for thinking about how students learn – and are taught – programming languages, and supports a compelling argument for investigating pedagogies that address language acquisition factors. I've previously described supplemental instructional strategies that focus on the teaching of programming languages as languages per se, and that, importantly, reach grade-level students who formerly would have learned little (Portnoff, 2016 / 2018). I therefore contend that the root cause of the NPFP is a pedagogic gap – we want students to use logic to solve problems, but we neglect to provide instruction that will help them acquire the very language which mediates that logic.

Human language is an innate, highly specific cognitive ability quite distinct from general intelligence. A language can't be taught explicitly; for example, through grammar instruction. Rather teachers need to provide the conditions, situations, and experiences that facilitate its gradual acquisition implicitly through: (a) repetitive exposure to language data – vocabulary, syntactic patterns, and paradigms – in meaningful contexts; and (b) intentional expressive use of the language with implicit feedback. For most students, the start of meaningful automaticity and basic programming fluency – like the timeline for foreign languages – emerges after two to three years. Expectations for acquisition of programming skills therefore need to be adjusted and the timeline extended from a single course to a multi-year course sequence.

Next time: Supplemental instructional strategies that can help students acquire programming languages implicitly. (HW)

Scott teaches Computer Science at Downtown Magnets High School in the Los Angeles Unified School District. In 2013, he wrote a contextualised introductory programming course that uses models and simulations for exploring real-world cross-curricular topics spanning the fields of art, geography, astronomy, political science, and biology, into which he is currently integrating language acquisition strategies. A unit from this course, Dynamic Word Clouds, resides in the Engage-CS-Edu repository and earned an Engagement Excellence designation.