BLOG @CACM

The *Communications* Web site, <http://cacm.acm.org>, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

Follow us on Twitter at <http://twitter.com/blogCACM>

DOI:10.1145/2994590

<http://cacm.acm.org/blogs/blog-cacm>

Introducing CS to Newcomers, and JES As a Teaching Tool

Valerie Barr gets high schoolers thinking about CS, while Mark Guzdial mulls the benefits of Jython Environment for Students.

Valerie Barr
A Very Local Snapshot of K-12 CS Education

<http://bit.ly/2cqfRDg>
June 30, 2016

I had an interesting experience recently. I agreed to run a session on computer science for the STEP (Science and Technology Entry Program) students at Union College's Kenney Community Center. The range of students was large, from 7th to 12th grade. Usually in a session like this I start by asking two things. First, in what ways are computers already in their lives? Second, what kind of problems do they think computers can help solve? There were lots of interesting answers to both questions, and I am always intrigued there are many computers in their lives that kids (and adults) do not think about. It is also fun to get sidetracked by discussion about user interfaces—go heat something up in your microwave, for example, and contemplate the fact that you can run it for 99 seconds, but 100 gets you a minute!

My overall goal is to try to get the kids to think about problems that are not obviously numeric in nature. This allows us to talk about the dual challenge of figuring out a solution to the problem and then figuring out how to get the “data” into the computer. Of course, I want things to be fun too! I have two aids I typically use: SET (a visual perception card game) and 3D Squares matching puzzles (unfortunately, no longer available for purchase). These lead us to talk about how a human might solve a problem versus what we would have to tell the computer to do. Then there is that pesky question about how to get the SET cards or the puzzle pieces “into” the computer. This is where things got interesting.

I had asked the students, as they introduced themselves, to tell me what sort of prior experience they had had with computers. Just about everyone one had done something formal either in school or in a camp setting. Several of them had programmed in Python, at least one knew some Java, and several had done robotics work, but as a group,

none of them seemed to have learned anything about data representation inside the computer. Furthermore, though many of them were gamers, that experience did not seem to keep them from being surprised that you could come up with ways to represent cards or puzzle pieces inside the computer in ways that would allow a program to reason about those entities in the same ways that a human might. Certainly I do not expect K-12 CS education to get deep into mechanisms for data representation, but I do expect that today's students, who as a cohort carry around devices that hold audio files, image files, and video files, should be a bit more facile with the idea that other entities, like cards and hard plastic puzzle pieces, could also be represented in some way inside the computer.

We wrapped up with a quick overview of Traveling Salesperson (always fun to ask kids where in the world they would like to go) because I like to prove to them that, as amazing as computers are, there are still problems that computers cannot solve. That concept also seemed quite surprising to them.

As a snapshot of the state of K-12 CS education, my experience was not particularly encouraging, but it does make a strong case for both the coherent curricula and the formal professional development on which many in the CS community are working. I am ever optimistic that as I continue to do these sorts of sessions, I will begin to see evidence of the impact of new curricula and better-trained teachers. Stay tuned!

Mark Guzdial
14 Years of a
Learner-Centered
Python IDE

<http://bit.ly/2aNXAnC>
 August 10, 2016

I recently discovered that the latest version of our Python IDE for learners, JES (Jython Environment for Students), passed 10,000 downloads (see the count at <http://bit.ly/2cyt1l2>); 10,562 when I started writing this post. We created JES in 2002 in support of our Media Computation course at Georgia Tech, which is a required course for students in our colleges of Liberal Arts, Business, and Design. Schools that adopted our Python Media Computation textbook have also mostly adopted JES, or used options that support the same libraries, such as Pythy or Pyjama. The current download count is probably an underestimate of users, since some schools download JES only once and then distribute it across campus.

Most student Python programmers use IDLE, the integrated development environment (IDE) provided with Python. While Python is more learner-centered than many languages used by professionals (see “Five Principles for Programming Languages for Learners,” <http://bit.ly/2chERzG>), the IDE should also be tailored for learners. Learners have different goals and needs than professional programmers. They need programming environments that take a learner-centered design approach (<http://bit.ly/2cEcQnz>), like JES or DrPython.

JES is likely one of the most-used pedagogical programming environments for Python, and the fact that it is still frequently used at the ripe old age of 14 suggests that it has been pretty successful. It is probably worth considering why it has worked. As lead on the team that built it and maintained it for the last 14 years, I am not a good judge for why it has worked.

Instead, I offer four brief stories about JES’s development and maintenance over the last 14 years.

Keep It Simple: From DrScheme and DrJava, we took the principle to keep it simple. We did not want an interface with many options for many kinds of uses. We wanted an interface that worked very well for learners.

In JES, you can only edit one file at a time, with an interaction pane (a REPL, <http://bit.ly/2cEcZrl>) for testing and running code. Most Python programs by professionals use multiple files. We opted to design for the beginner who could get lost in a sea of files. Having more than one file open requires some interface for switching files. One file means no additional interface. You can build bigger things in JES, but we optimized for the most common learner case.

Imagineering Media Computation to be Normal Python: We built JES to facilitate students programming in Media Computation. Students can program anything in JES; it is a full Python implementation. We added additional features to JES to facilitate Media Computation with a minimum of cognitive load.

We imagineered a community of practice around Media Computation (a story we told in <http://bit.ly/2bRCSmd>). In JES, it is normal for Python to know how to make pictures with `makePicture`, access individual pixels with `getPixels`, change colors with `setRed`, and access sound samples with `getSamples`. Invisibly, we load libraries for the student so that JES is a Python that supports programming with multimedia from the first moment of class. In normal Python, you can `print` anything to get its value. In JES, you explore any picture or sound to open an explorer to see color values in pictures or to visualize the samples in a sound.

Later on, we tell students there are libraries and how JES is automatically loading them. But on Day One, JES is a friendly Python that knows about pixels and pictures, sounds and samples, and frames and videos. No imported libraries, no dots, no extra details.

Sometimes, you explain errors after they occur: When we first started teaching Python to non-technical majors at Georgia Tech, students struggled with indentation errors. Even today in our ebooks (<http://bit.ly/2bW119J>), indentation problems in Python are among the most common and the most difficult for students to fix.

We decided to add a small feature to JES to help with indentation errors. We wanted indentation to be salient and obvious. JES draws a small

blue box around the lines in the same block as the current line (containing the cursor).

Now, when students come to me with indentation errors, I ask them “Where’s the blue box?” And they invariably ask, “*What* blue box?” I point it out to them. “Ohhhh—that blue box! Yeah, that’s useful.” Until students realize they have to attend to indentation, they do not see the support we are providing. *Telling* them that indentation is important and pointing out the blue box is not nearly as effective as encountering the error once.

Over the years, remove features: The original 2002 JES had more features in it than the most recent version. Two examples:

- We used to have a menu item to automatically turn in homework. It worked for a while at Georgia Tech, but then we changed how we handled homework. Other schools have always had their own mechanisms, and most schools try to make it easy to turn in homework. We dropped that.

- We worried about students loading a function from their homework file, then deleting the source code (by accident). The program might work correctly for them, but the file would be incomplete when grading. So, we wrote complicated code that compared the internal namespace to the source code, but that complicated code often caused problems of one sort. We decided it was not worth a big maintenance task for a rare error case, so we simply removed all of that complicated code.

The lesson here is Yogi Berra’s famous quote, “It’s tough to make predictions, especially about the future.” Some of the issues that seemed critical in the beginning simply were not all that important in actual practice later. The features with more staying power were the ones that we added in response to learner behavior. Our most successful features were the ones we added to meet real learners’ needs, not what we initially imagined the needs to be.

Valerie Barr is a professor in the Computer Science Department at Union College, Schenectady, NY.

Mark Guzdial is a professor in the College of Computing at the Georgia Institute of Technology, Atlanta, GA.

© 2016 ACM 0001-0782/16/11 \$15.00