Using Python in a High School Computer Science Program

Jeffrey Elkner
Yorktown High School, Arlington, Virginia

December 6, 1999

Abstract

Yorktown High School, in Arlington Virginia, has begun a large scale experiment using the Python programming language in a high school computer science environment. Motivated by the need to find an alternative to C++, which was not meeting the needs of our first year students, and inspired by Computer Programming for Everybody, we are using Python with all levels of students, from our beginning programmers to our most experienced. We are using Python both as an instructional and productivity tool, and we are helping to develop the educational resources needed to make the use of Python practical in a school environment.

How and Why We Came to Use Python

In 1999, the College Board's Advanced Placement Computer Science exam was given in C++ for the first time. The 1996 Advanced Placement Course Description for Computer Science had this to say about the new language choice:

> *The committee recognizes that no single programming language satisfies all needs ideally. However, C++ provides a standard mechanism for emphasizing and implementing modularity and abstraction, which are key concepts in introductory computer science. Moreover, students completing an AP course using C++ are likely to receive credit or placement at a wide range of colleges [College Board].*

As in many high schools throughout the country, the decision to change languages had a direct impact on the computer science curriculum at Yorktown High. Up to this point, Pascal was the language of instruction in both our first year and AP courses. In keeping with our past practice of giving our students two years of exposure to the same language, we made the decision to switch to C++ in our first year course for the 1997-98 school year, so that they would be in step with the College Board's change for the AP course the following year.

Two years later, I was convinced that C++ was a poor choice to use for introducing students to computer science. While it is certainly a very powerful programming language, it is also an extremely difficult language to learn and teach. I found myself constantly fighting with C++'s difficult syntax and multiple ways of doing things, and I was losing many students unnecessarily as a result. Convinced there had to be a better language choice for our first year class, I went looking for an alternative to C++.

A discussion on the High School Linux Users' Group mailing list provided me with a possible solution. A very exciting thread emerged during the latter part of January, 1999 concerning the best programming

language for use with first time high school computer science students. In a posting on January 30th, Brendon Ranking said:

> *I believe that Python is the best choice for any entry-level programming class. It teaches proper programming principles while being incredibly easy to learn. It is also designed to be object oriented from its inception so it doesn't have the add-on pain that both Perl and C++ suffer from..... It is also *very* widely supported and very much web-centric, as well [Ranking].*

I had first heard of Python a few years earlier at a Linux Install Fest, when a very enthusiastic Michael McLay told me about Python's many merits. He and Brendon had now convinced me that I needed to look into Python as the possible solution to our first language problem. As a high school teacher with all the obligations that entails, however, I was not in a position to drop everything and give myself a crash course in Python.

Unable to check out Python myself, I did what I often do when there is something important to learn and I am not able to learn it; I asked one of Yorktown's many bright young minds to check it out for me. Matt Ahrens was a student who I had introduced to Linux a few years before, and who in two years had gone from being my student to my teacher on all matters relating to Linux. He jumped at the chance to try out Python, and in the final two months of the school year he not only learned the language but wrote an application called pyTicket that allowed staff at our school to use the web to report technology problems by creating "tickets" which student volunteers could handle. I knew that it would not have been possible for Matt to finish an application like that in C++, and this accomplishment combined with Matt's very positive assessment of Python brought us to the next step in the investigation.

Matt wrote to Guido Van Rossum, who soon told us about a very exciting project underway at CNRI to develop materials which would use Python to make "Computer Programming for Everybody" a realistic possibility [Van Rossum]. It was the existence of this project that finally convinced me to use Python the following year. The one concern I had once I started investigating Python was the apparent lack of resources that would be available to me using Python in a classroom setting. There was no text book for starters, nor were there lesson plans, teaching guides, or any of the other teaching materials to which teachers are accustomed.

The prospect of "Computer Programming for Everybody" (CP4E) meant that help would be available, and turned what might have otherwise been an insurmountable obstacle into an opportunity. The sixth paragraph of CP4E lists its three components:

- Develop a new computing curriculum suitable for high school and college students.
- Create better, easier to use tools for program development and analysis.
- Build a user community around all of the above, encouraging feedback and self-help.

The first two components held out the promise that the missing tools would soon be forthcoming, and the third component offered the opportunity to be an early entrant into this new user community as well as the challenge to begin the "self-help" that would make the community viable.

Finding a Text Book: Help and Self-help

Having decided to use Python in both of my introductory computer science classes the following year, the most pressing problem was the lack of an available text book. I had taught without a text book before, and actually enjoyed doing so, but given the general lack of

entry level resources in Python it did not seem prudent to go forward without something that students could use to support what they were learning in class.

The solution to this problem came in the form of help that would in turn require self-help. Earlier in the year Richard Stallman had introduced me to Allen Downey, a computer science professor at Colby College. Both of us had written to Richard expressing an interest in developing free educational content. Allen had already written a first year computer science text book, How to Think Like a Computer Scientist [Downey]. When I read this text book I knew immediately that I wanted to use it in my class. It was the clearest and most helpful computer science text I had ever read. It emphasized the processes of thought involved in programming, rather than the features of a particular language. Reading it immediately made me a better teacher.

Not only was "How to Think Like a Computer Scientist" an excellent book, but it was also released under the GNU General Public License, which meant it could be used freely and modified to meet the needs of its user. I set about creating an on line version of it, and while writing to Allen about that project, he told me that he would be doing an AP C++ version over the following summer. I did not hesitate in telling him that I would use it in my Advanced Placement Computer Science class.

Once I decided to use Python, it occurred to me that I could translate Allen's book into the new language. This would offer yet another exciting opportunity. While I would not have been capable of writing a text book on my own, having Allen's book to work from would make it possible for me to do so, at the same time demonstrating that the cooperative development model used so well in software would also work for educational content.

I have currently completed translating six chapters. The book is available on line at http://yhslug.tux.org/obp/thinkCS/thinkCSpy and I have been using it with my introductory computer science classes as planned. I have been writing each chapter in time to use it in class in what might be called a "just in time" translation system ;-) It is apparent that my students enjoy using the book. Many of them have expressed to me how much they like both the C++ version and the new Python one. Having it on line has the added advantage that I can make instant changes whenever a student finds a spelling error or difficult passage. I encourage students to look for errors in the book by giving them a bonus point every time they find or suggest something that results in a change in the text. This has the double benefit of encouraging them to read the text more carefully, and of getting the text thoroughly reviewed by its most important critics, the students who will be using it to learn computer science.

Into the Classroom: Preliminary Thoughts on Python as a High School Programming Language

From the process of translating and using "How to Think Like a Computer Scientist", I can already reach some preliminary conclusions about Python's suitability to teaching beginning students. The first and most obvious conclusion is extremely encouraging: Python greatly simplifies programming examples and makes important programming ideas easier to teach.

The first example from the text dramatically illustrates this point. It is the traditional "hello, world" program, which in the C++ version looks like this:

```
#include <iostream.h>

\\ main:generate some simple output

void main()
{
  cout << "Hello, world." << endl;

}
```

and which in the Python version becomes:

```
print "Hello, World!"
```

Even though this is a trivial example, the advantages to Python stand out. There are no prerequisites to Yorktown's Computer Science I course, so many of the students seeing this example are looking at their first program. Some of them are undoubtedly a little nervous, having heard that computer programming is difficult to learn. The C++ version has always forced me to choose between two unsatisfying options: either to explain the `#include`, `void main()`, `{`, and `}` statements, and risk confusing or intimidating some of the students right at the start, or to tell them "just don't worry about all of that stuff now, we will talk about it later" and risk the same thing. The educational objectives at this point in the course are to introduce students to the idea of a programming statement and to get them to make their first program, thereby introducing them to the programming environment. The Python program has exactly what is needed to do these things, and nothing more.

Comparing Section 1.5 of each version of the book, where this first program is located, further illustrates what this means to the beginning student. There are thirteen paragraphs of explanation of "Hello, world" in the C++ version, in the Python version there are only three. More importantly, the missing ten paragraphs do not deal with the "big ideas" in computer programming, but with the minutia of C++ syntax. I found this same thing happening throughout the chapters that I have completed so far. Whole paragraphs simply disappear from the Python version of the text because Python's much simpler syntax renders them unnecessary.

Teaching with Python is enabling me to rethink many of the strategies I have used for the last several years. Because C++ is a middle level language, I have started the year with a three week introduction to binary arithmetic. So much of C++ is close enough to the hardware of the machine that I found this to be the only way I could prepare students to understand what is going on in their programs. I began the current year with the same binary unit, but I am now realizing that with Python it is completely unnecessary to do so. A low level understanding of how computers work is very important for computer science students, and I still plan to teach binary arithmetic next year. Instead of giving up those most valuable beginning weeks of the year to it, however, and putting up a hurdle for students to jump before they can begin to learn programming, I will move it to the last quarter of the year, when I plan to introduce students to C, and at which time they will already be comfortable with programming.

Using a very high level language like Python allows a teacher to postpone talking about low level details of the machine until students have the background that they need to better make sense of the details. It thus creates the ability to put "first things first" pedagogically. The clearest example of this is the way in which Python handles variables. In C++ a variable is a name for a place which holds a thing. Variables have to be declared with types at least in part because the size of the place to which they refer needs to be predetermined. Thus the idea of a variable is bound up with the hardware of the machine. The powerful and fundamental concept of a variable is already difficult enough for beginning students (in both Computer Science and Algebra). Bytes and addresses do not help the matter. In Python a variable is a name which refers to a thing. This is a far more intuitive concept for beginning students, and one which is much closer to the meaning of variable that they learned in their math class. I had much less difficulty teaching variables this year than I did in the past, and I spent less time helping students with problems using them.

Another example of how Python aides in the teaching and learning of programming is in its syntax for functions. Of all the things that I learned by using Python this year, the way in

which the right tool could help in explaining functions was the most exciting. My students have always had a great deal of difficulty understanding functions. The main problem centers around the difference between a function definition and a function call, and the related distinction between a parameter and an argument. Python comes to the rescue with syntax that is nothing short of beautiful. Function definitions begin with the key word `def`, so I simply tell my students, "when you define a function, begin with `def`, followed by the name of the function that you are defining, when you call a function, simply call (type) out its name." Parameters go with definitions, arguments go with calls. There are no return types or parameter types or reference and value parameters to get in the way, so I was able to teach functions this year in less then half the time that it usually took me, with what appears to be better comprehension.

One more feature of Python syntax which ought to be discussed is Python's use of indentation to determine program structure. I had expected this to be a great help in facilitating the teaching of well formatted, readable programs. Indeed, it has not disappointed in this regard. Python's syntax makes programs much easier to read, thereby making it easier to effectively evaluate student work and to identify problems in student programs. This is a big plus for teachers using Python in the classroom. Increased readability also enables students to work together on larger and more complex programs. Lex Berezhny mentions how this has helped with the two projects on which he is working in the appendix which follows this paper.

I was caught a bit by surprise, however, by the way that the absence of explicit begin and end statements has made it more difficult for beginning students to understand the idea of a code block. There has been more problems then usual with students coding infinite while loops simply because the increment statement is outside the body of the loop. The idea of several statements acting as one takes some getting used to, and it appears the visual cues of a begin and an end make it easier to understand. This is only a temporary difficulty, however, and one more then compensated for by the benefits that increased readability and shorter programs will continue to bring long after the problem understanding code blocks has been forgotten.

Translating "How to Think Like a Computer Scientist" has been a valuable educational experience for me, and so far it has been a largely straightforward process. I have been able to translate all of the sample code through the first six chapters directly into Python without changes to either the structure or logic. The Python code is shorter, easier to read, and much "prettier" to be sure, but other than that it is the same.

All that is about to change, however, and a brief look ahead reveals some challenges. The next chapter, 7, discusses strings, one of the uglier topics in C++. This chapter will be so much shorter in Python that I am planning on discussing general Python sequences in addition to strings. After that there are two chapters on C++ structures. I do not want to introduce classes at this point, but there is nothing else in Python that will allow for a named thing with named parts, so I will either have to change the way the example programs are written or come up with entirely new examples.

This is both exciting and a bit frightening for me. Up to now I have been relying on Allen Downey for pedagogy. While I have made some minor changes in emphasis to accommodate my less experienced students, the ideas in the text are his. I plan to continue to do that as much as possible, but differences between the languages will now require more substantial changes.

There is a related issue concerning whether or not Python's different way of doing things will adequately prepare students for the C++ they will be learning if they go on to the

Advanced Placement course the following year, or that they will likely be using in college. This is both a practical concern for the success of the students learning with Python, as well as a political concern in getting Python accepted into high schools. Test scores are an import measure of the success of high school programs, with Advanced Placement exams being among the most import of these. It will need to be established early on that learning Python in no way harms AP test results if Python's use is to be widely accepted.

Fortunately, my experience even at this early stage leads me to believe that learning Python will actually improve the effectiveness of our computer science program for all students. I can already see a higher general level of success and a lower level of frustration from my students than I experienced during the last two years using C++. I am moving faster with better results. More students will leave the course this year with the ability to create meaningful programs, and with the positive attitude toward the experience of programming that this will generate. Because we are able to move faster with Python, I fully expect to be able to teach students to use classes by the early Spring. This will give students going on to AP Computer Science the framework for better understanding things like structures and classes in C++, and will thus improve their performance in that important subject.

Finally, I am also using Python with a number of Yorktown's more advanced students in a variety of different ways. Like the proverbial man with a hammer who sees every problem as a nail, my excitement about Python makes me continually devise new uses for it. Luckily, this is turning out to be both fun for the students and consistent with the goal of investigating Python's applicability to our computer science program.

Using Python With Our Advanced Programmers

Yorktown offers a course called Computer Information Systems Advanced Topics, which is providing an opportunity to look at different ways that using Python could benefit high school CS programs. This course is offered to students who have completed the first year programming course and who want a further opportunity to study computer science outside the AP course. Several of these students have already completed AP Computer Science, and others are enrolled in it concurrently. The ways that I am using Python in this course fall into two broad categories: using Python as an aide to understanding the AP curriculum, and using Python as a development tool for our most advanced programmers.

Two of our students, Louie Corbo and Vipul Sharma, are using Python in Advanced Topics as a way to better understand their assignments from AP CS. The idea is that seeing the same problem solved in two different languages will deepen their understanding of the underlying algorithms involved. Also, Python's greater simplicity will make the solutions easier to understand. They have just begun doing this, so it is too early to report on the results.

Three of Yorktown's top young programmers, Lex Berezhny, Jonah Cohen, and Stefan Wrobel, are using Python to develop application programs which we will use at our school. An appendix to this paper includes a description by Lex of the two main projects on which they are currently working. The idea here is to give students experience working together on a large project in an environment which resembles as much as possible the kind of environment that they are likely to encounter as professional programmers. Nothing could be better in this respect than writing something which will actually be used.

Again, it is too early analyze results, but one of programs, which we call pyTicket, is already in use. This program implements a web based interface for processing requests by our staff for technology assistance. The technologies employed by this program could be applied to solve several other similar problems with tracking information in the school. The other program, Student Portfolio, if successful, could be adopted by the school system for use system wide. We are currently considering getting other students involved in these and similar projects.

Where To From Here? Opportunities and Obstacles

Python has already demonstrated that it is well suited as a programming language for beginning programmers. My experience using it at Yorktown confirms this. The possibility now exists for being able to successfully teach larger numbers of students the basic ideas of computer science. With the growing interest on the part of school systems in all things related to computer technology, this is a good time to begin using Python in schools.

The greatest obstacle, however, to using Python to teach high school students programming is the almost complete lack of educational resources. Python's success in being accepted by high school CS departments and teachers will depend on the availability of these educational support materials. The current initiative underway at CNRI offers the promise that the needed resources will soon be forthcoming, and that "Computer Programming for Everybody" will move from being an idea whose time has come to being a reality.

References

[College Board] College Entrance Examination Board and Educational Testing Service (1995), "Advanced Placement Course Description: Computer Science", p4

[Downey], Allen B. (1999), "How to Think Like a Computer Scientist" http://www.cs.colby.edu/~downey/ost (accessed 11/18/99)

[Rankin], Brendon (1999), "Re: [HS-LUG] Ok, I'm willing to go there", Discussion on the HS-LUG mailing list. http://hs-lug.tux.org/archives/hs-lug-digest.19990130 (accessed 11/13/99)

[Van Rossum] Guido, "Computer Programming for Everybody" http://www.python.org/doc/essays/cp4e.html (accessed 11/17/99) Corporation for National Research Initiatives (CNRI)